"Smart Cafe": A Mobile Local Computing System Based On Indoor Virtual Cloud

PU Lingjun, XU Jingdong, YU Bowen, ZHANG Jianzhong

College of Computer and Control Engineering Nankai University, Tianjin, 300071, China

Abstract: With network developing and virtualization rising, more and more indoor environment (POIs) such as cafe, library, office, even bus and subway can provide plenty of bandwidth and computing resources. Meanwhile many people daily spending much time in them are still suffering from the mobile device with limited resources. This situation implies a novel local cloud computing paradigm in which mobile device can leverage nearby resources to facilitate task execution. In this paper, we implement a mobile local computing system based on indoor virtual cloud. This system mainly contains three key components: 1) As to application, we create a parser to generate the "method call and cost tree" and analyze it to identify resource-intensive methods. 2) As to mobile device, we design a self-learning execution controller to make offloading decision at runtime. 3) As to cloud, we construct a social scheduling based application-isolation virtual cloud model. The evaluation results demonstrate that our system is effective and efficient by evaluating CPU-intensive calculation application, Memory-intensive image translation application and I/O-intensive image downloading application.

Key words: mobile local computing system, application partition, dynamic offloading strategy, virtual cloud model, social scheduling

I. INTRODUCTION

Nowadays, with the rapid development of mobile devices and the increase of colorful mobile applications, more and more people regard mobile devices as a part of their lives. They can use them to pay for ticket, book hotel, play games, and etc. However, these devices still suffer from limited resources especially energy and computing resources. For example, the battery power soon runs out when playing games or watching videos and it takes a long time to beautify High-Definition landscape picture [1].

In these years, with CPU and storage hardware reforming, Wi-Fi technology developing and virtualization rising, many indoor environment such as cafe, library, office even bus and subway has deployed some facilities to provide plenty of bandwidth and computing resources, which infers that the local cloud has great potential to play the role of public cloud in mobile cloud computing. For instance, Satya [2] proposes a 3-layer Cloudlet in which the device could first register on the nearby Virtual Machine (VM) and then that VM regarded as surrogate could further fulfill some tasks for device. They propose the idea while do not mention the specific scheme. As an answer to this issue, Chun [3] proposes the CloneCloud in which the device OS would be completely cloned in the VM and the device could dynamically offload the heavy-tasks to execute in its counterpart. However, this model could not adapt to indoor environment (POIs) because nobody wants to clone their devices wherever they visit. In this paper, dif-

In this paper, we implement a mobile local computing system based on indoor virtual cloud.

ferent from CloneCloud requiring cloned OS, we aim to implement a device-independent offloading system.

Since many citizens will move around at limited number of POIs and will spend more than 70% time in those POIs [4], we implement a practical and meaningful method offloading computing system based on indoor virtual cloud in POIs where customers could enjoy extra resources and POIs could appeal much more customers. This system contains the following three crucial components as shown in figure 1:

1) As to mobile application, we create an automatic encapsulation tool including a parser to generate the "method call and cost tree", an analyzer to identify resource-intensive methods and a translator to decorate those methods.

2) As to mobile device, to effectively offload those components to the cloud, we design an execution controller including a status profiler to collect the current device status, a XML parser to require and update the resource consumption of method and with these results a decision maker is designed to achieve better Quality of Service (QoS) by an energy priority scheme.

3) As to cloud, to concurrent support large scale offloading service efficiently, we construct a social scheduling based application-isolation virtual cloud model in XEN virtualized platform [16].

II. Related Work

The basic idea of mobile cloud computing is to offload heavy tasks from resource-limited device to resource-sufficient cloud. Existing approaches generally fall into two broad categories: thread-based offloading and method-based offloading. The promising way in the first category is CloneCloud [3] in which a completely copy of device OS is resident in public cloud. Device transfers soft and hard states to run heavy tasks in the cloud. Based on this framework, COMET [5] proposes a thread-based offloading scheme in which device can transparently transfer the running thread to the cloud. Although thread-based offloading is no need to understand or modify the applications, thread migration will incur high shipping overhead and this strategy does not work for indoor environment (POIs) where it is impractical to deploy the cloned device OS for each user in advance. Our offloading system comes under the latter category which is totally device-independent. Examples in this category include Xray [6], MAUI [7] and ThinkAir [8] in which applications are explicitly written for local and remote executions. This approach not only incurs less offloading overhead (method reflection mechanism) but also is independent to device OS.

As to locate resource-intensive methods, Maui and ThinkAir leverage the C# (Java) annotation to mark the "offloadable" methods and dynamically construct the offloading flow at runtime. They need to modify the compiler or library in the smartphone to facilitate the offloading and do not mention how to identify the resource-intensive methods. Xray implements a dynamic way to identify "offloadable" methods by tracing all the relevant system-level and application-level events during application execution without developer awareness. It regards them as those never trigger any hardware-related methods. However, this approach will put the methods with little resource consumption in "offloadable". To overcome these shortcomings, our approach is to unleash the burden (e.g. identify resource-intensive methods) to the application developers who have more knowledge about the application. We do not modify the compiler or library in the smartphone but design a developer framework to assist the developers to identify the resource-intensive parts and to construct the offloading flow offline.

As to effectively offload those methods to the cloud, MAUI and ThinkAir dynamically obtain the current status such as CPU & Memory utilization, network condition (RTT) and residual energy associated with input parame-

ter to make runtime decision. Similar ideas [9] leverage bandwidth and energy cost to make offloading decision. In theory, they can be optimal. But they may not only introduce high overhead (network condition estimation) but also not be flexible to different kinds of input (calculation parameter or image size). In this paper, we believe the application developers can supply many experience data saved as an xml file since they need to check the application with different parameters in testing stage. With this xml file we propose a lightweight offloading decision strategy and we regard the "CurVal" in xml file can well reflect the network or service congestion, which avoids estimating the network condition before every offloading decision.

As to concurrent support offloading service efficiently, different from traditional FIFO or delay priority scheduling, we propose a social priority scheduling which means the users with higher relation to the POIs can get better services. And we implement an application-isolation virtual cloud model in XEN platform, which is not considered by the preceding method-based offloading systems. Overall, the main contribution of our system is three folders. First is to design a Developer Framework which can identify the resource-intensive parts and reconstruct the application offline. Second is to implement a lightweight offloading decision strategy. Third is to design a social-based virtual cloud model. It can well support concurrent service from the users, which is different from some previous researches [10, 11].

III. Motivation

Intuitionally, if cafes or POIs always give some discount, it can appeal more customers. But what if they provide more bandwidth and computing resources? To answer this issue, we make a questionnaire including Q1. Would you like to visit our Uni-cafe? Q2. If it provides extra bandwidth and computing resources would you like to visit? Q3. If it provides those resources but double the price would you like to visit? To avoid inference among them and to ensure randomness, we random walk in our university to ask one student one random question.

In nearly 200 answers to each question, 38% would like in Q2 (30% in Q1, 5% in Q3) and more than 70% in Q2 and Q3 concern what and how the extra resources can help them. These results imply that people are interested in leveraging surrounding resources and the cafe providing them can appeal more attention (although price still dominates). Therefore we aim to design a "Smart Cafe" which can not only leverage local cloud resources to achieve better QoS but also provide advanced services according to social profile with "cafe" (POIs) such as visit frequency, residence time and shopping bill so as to appeal more new customers and retain old customers.

Generally speaking, the typical applications in indoor environment would be gaming, chatting, web surfing, image/video editing and etc. The resource-hungry tasks in them can be categorized as CPU-intensive (some calculation in gaming), Memory-intensive (image or video editing) and I/O-intensive methods (resource downloading from web). In this paper, we aim to identify these burden and use the cloud resources to facilitate them. Meanwhile, it is difficult to unify the user demands in a POI and different POIs may also share different demand preference. So we aim to propose a general offloading framework for all three categories rather than a specific one.

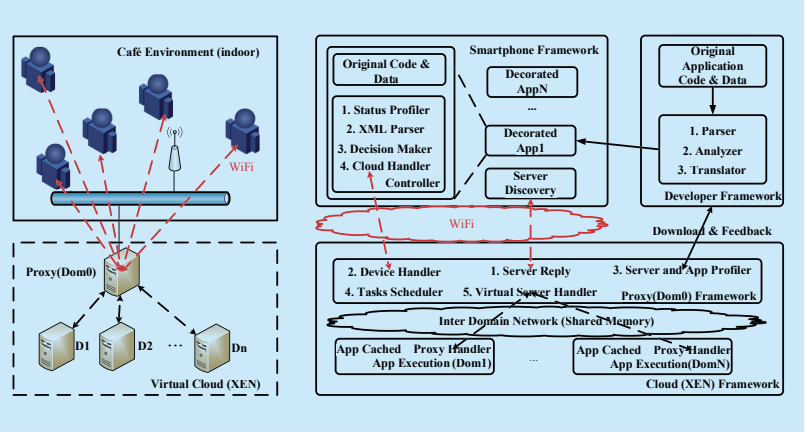

Fig.1 *Physical environment and our proposed system architecture*

IV. Architecture Overview

Our modular-based mobile local cloud computing system as depicted in figure 1 contains three parts: Developer Framework, Device Framework and Local Cloud Framework.

Frankly speaking, current applications cannot directly adapt to our mobile cloud computing system. In order to ease developers and leverage current apps, we design an automatic encapsulation tool for Android Apps. Specifically, once developers put source codes associated with the execution profiles generated by the android "traceview" into our tool, the *Parser* first constructs the "method call and cost tree" as depicted in figure 3. Then the *Analyzer* leverages a down-top analysis to identify and mark which methods should be offloaded. Finally, in terms of the analysis results, the *Translator* decorates the original source codes with some accessories (i.e. execution controller) for offloading execution. The decorated application can well support our system.

When it comes to Device Framework, in order to make offloading smoothly, we first use the "broadcast" to find out the Proxy (*Server Discovery*). Then as the application runs, the execution controller including a *Status Profiler*, *XML Parser*, *Decision Maker* and *Cloud Handler* will be triggered for each "offloadable" method. It first collects the device status (e.g. the residual energy) and obtains this method resource consumption according to its input by querying the XML record. Next it adopts a simple energy priority algorithm to make offloading decision. When the method is decided to offload, the cloud handler will send the method profile associated with the social profile to the local cloud and retrieve the result to continue following execution.

Local Cloud Framework is built on XEN virtualized platform which is master (Dom0)-workers (DomU) model in nature. When the *Device Handler* receives the method profile and social profile, it first checks the *Server and App Profiler* whether and which Domain can support it, then according to the social profile distributes extra resources to corresponding worker Domain and throws this task to social-priority *Tasks Scheduler*. When it is turn to run that task, the *Virtual Server Handler* will send the task (method profile) to corresponding Domain and finally the task can be executed. If the task cannot be supported currently, the Device Handler will ask for the .apk file from device if its social profile is preferred. Then the local cloud will save it in Server and App Profile and assign a designated worker Domain for it. From then on, the task can be offloaded execution smoothly. The general execution flow is shown in figure 2.

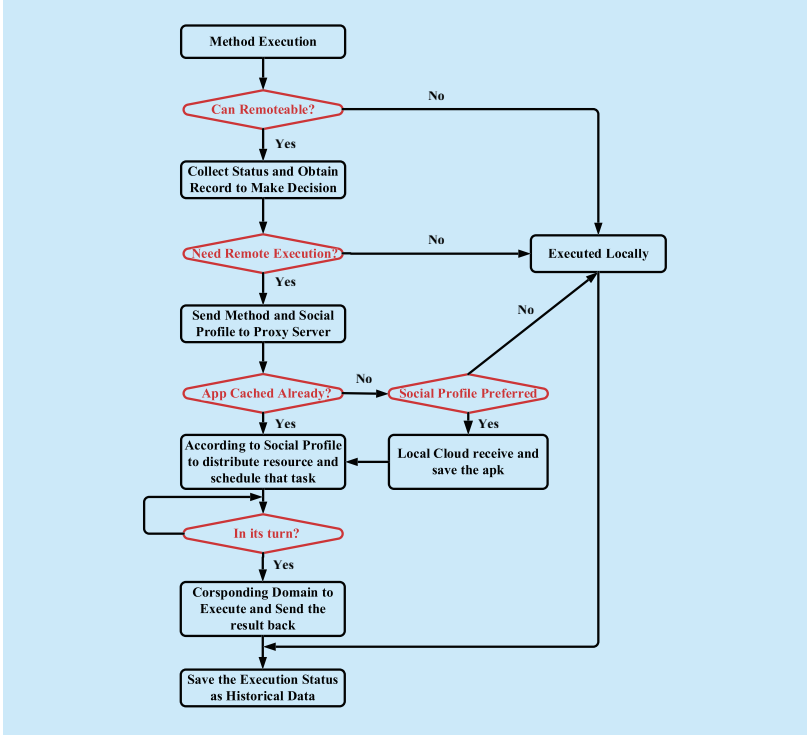

Fig.2 *Method execution flow*

V. Component Implementation

5.1 Developer framework

Recently, an increasing number of mobile developers have focused on the Android platform. For example, nearly one thousand applications were launched in Android Market every month in 2013. Therefore, our encapsulation tool is designed for Android application to meet the huge potential needs.

Different from Saarinen [12] who proposed

a tool to identify communication-related heavy methods, we consider computing-related heavy methods. In order to identify them, we need obtain the "method call and cost tree", a directed weighted graph where the vertex represents method, the edge represents calling relationship and the edge-weight represents the resource consumption ratio. Since the android analysis tool "traceview" can generate application trace file containing the calling relationship and record method execution time (exclusive the sub-methods execution time), we create a parser to extract the call and cost graph from the trace. To build up edge-weight, according to different input size, we respectively run the application in the device as well as in the cloud and record those trace files. (We assume the network is well connected and bandwidth is sufficient in local environment. So we omit the communication time). We regard the average of time consumption ratio between cloud and device as edge-weight.

Not all methods can be executed remotely such as the method that interacts with user interface or hardware. Therefore, if the name of one vertex contains "android/graphics" or "dalvik", this vertex and its ancestors cannot be offloaded. Meanwhile, not all methods need to be executed remotely. (e.g. the method that costs little resources). Based on these two principles, we create an analyzer with a down-top analysis. As depicted in figure 3, the analyzer scans the vertex name level by level to decide whether this vertex can be offloaded. If a vertex cannot be offloaded, the "offloadable" field of vertex will be set to false (the vertex marked slash). Then the analyzer traverses back up the graph from this vertex to the root (the dashed arrows) and regards these vertexes passed by also as "unoffloadable" ones. In addition, it proceeds to check the weight of the remaining vertexes. If the weight is lower than a threshold (set to 0.5), in other words the execution time in the cloud is twice less than that in the mobile device the vertex should be offloaded to the cloud. With this strategy, all the "offloadable" methods can be identified (Marked as dashed area). Finally, the analyzer adds the specific annotations to these methods and their corresponding classes.

After all "offloadable" methods identified, an application translator is constructed to meet the requirement of offloading system. This translator will be triggered to scan all the source files after the analyzer finishes analysis. Different from MAUI which use annotation to mark "offloadable method", we use it for translation. First, it checks whether the annotation '@ClassRemoteable' is in front of a class definition. And this mark means that there is at least one method needs to be executed remotely in this class. And then it further checks whether the annotation '@MethodRemoteable' is in front of method definition. When locating the "offloadable method", it extracts the method information to generate corresponding remote method. We will use an example to describe this process. Consider the following code:

```
@ ClassRemoteable
public class Calculate {
      private int result = 0;
      @ MethodRemoteable
      public int mul(int a, int b) {
   result = a*b;
  return result;
      }
  public int add(int a, int b) {return a+b;}
}
```

In class 'Calculate', 'mul' method should be offloaded and 'add' method does not. After code being translated, the Calculate class should like this:

```
public class Calculate implements Serializable{
      private int result = 0;
      public ExecutionController controller =
      ExecutionController.getInstance();
      public void CopyObject(Calculate c) {
          this. result = c. result;
      }
      public int mul(int a, int b) {
          Method toExecute = null;
          Class<?>[] paramTypes = {int.class,int.
class};
```

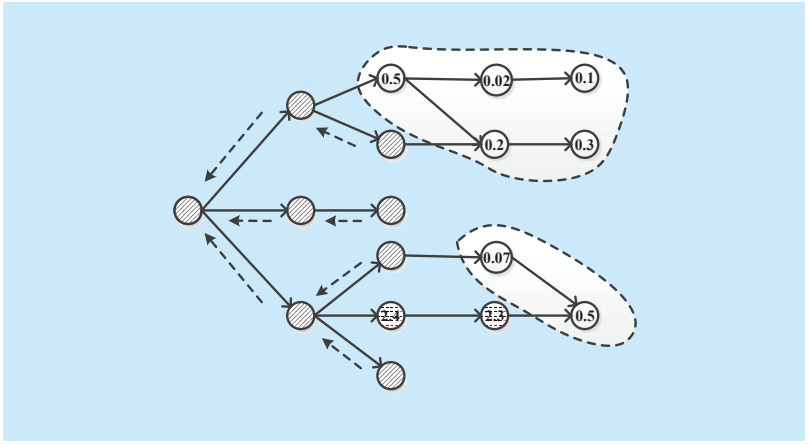

Fig.3 *Method Call and Cost Tree*

```
        Object[] paramValues = {a, b};
        Message msg = null;
        toExecute=this.getClass().getDeclaredMeth-
od
            ("localmul ",paramTypes);
        msg = controller.execute
            (toExecute, paramValues, this);
        if (msg.getThisObject() != null)
            CopyObject((Calculate) msg.getThisOb-
ject());
        return (int) msg.getResult();
    }
    public int localmul(int a, int b) {
        result = a*b;
        return result;
        }
        public int add(int a, int b) {return a+b;}
}
```

The original 'mul' method has been renamed as 'localmul'. That new name means it will run locally. And the new version 'mul' has the capability to run remotely by using java reflection mechanism. First it will get and transfer the information about 'localmul' to Execution Controller (in bold and italics) to decide whether this method should be executed remotely. (This translation approach can avoid method projection and runtime annotation checking in MAUI.) Wherever the method is executed the execution result and the object will be saved in the Message. After some assignments, the offloading execution completes. Meanwhile the translation (these attached parts) has little impact to the volume of whole application.

5.2 Device framework

In terms of finding available Proxy, we introduce multicasting mechanism (we assume the Proxy from different POIs will share a same unique multicast group and port). As soon as a mobile device accesses to WLAN of POI by Wi-Fi, Proxy Discovery Service will start to work. The Service packs the address of mobile device as data in the request and multicasts it. This request will be caught by the Proxy and the Proxy can extract mobile device's address. Then it unicasts back its address. At last, server discovery mission completes. The mobile device gets the Proxy address from response and the Proxy sets up a new thread to listen to this device.

To take full advantage of mobile cloud offloading system, making offloading decision is important. Prior works dynamically obtain the resource consumption of "offloadable" methods to make runtime decision. In this paper we use another "offload" idea which unleashes the "method-resource consumption" relation to the Developer Framework since the developers can run each application separately with different inputs and different network conditions in app testing phase (the same process of parsing "Method Call and Cost Tree"). In app testing phase, the method information including execution time (T), energy consumption (E) and corresponding input parameter are taken as historical data and saved in a private XML file for each application. With the help of developers, we implement an energy priority decision scheme:

1. energy=GetResidualBattery();
2. if energy < 20% **do** //energy decision
3. if $E_{remote} < E_{local}$ **do** //history data
4. return Offloading Execution;
5. if energy > 80% **do** //time decision
6. if $T_{remote} < T_{local}$ **do**
7. return Offloading Execution;
8. if energy in [20%, 80%] **do** // energy priority
9. if $(E_{remote}/E_{local})*0.7 + (T_{remote}/T_{local})*0.3 < 1$ **do**
10. return Offloading Execution;

Where E_{remote} represents the energy cost in smartphone when offloading that method to the cloud (mainly transmission cost) and E_{local} stands for the cost when that method running in the smartphone (mainly computation cost). To well evaluate them, we leverage the APIs of class BatteryManager from PowerTutor [14] to record the current energy before and after an "offloadable" method is invoked and the remainder represents the energy consumption which can achieve good precision (0.01J). In general, 20% (80%) power is taken as flag of low (high) power by most mobile devices. Here we take it as the threshold. The ratio between energy and execution time can be adjusted automatically based on the network condition. Ratio grows as network condition becomes better. In real world, it means that the better network condition, the more likely offloading which can save more energy happens. However, in real test we find different ratio has limited impact on the decision. This may because in ideal network condition offloading energy cost always far less than local energy cost of "offloadable" methods. In other words, the quotient of E_{remote} and E_{local} dominates the offloading decision. Since the E_{remote} also can well reflect the network condition, we fix the ratio as 7:3 in this paper.

In figure 1, once offloading decision maker obtains device status and "offloadable" method information, it will search for that method by scanning corresponding XML file. However, it is not sure whether the real input parameter can be found in historical data (it is impossible and useless to record all input cases). If it lies there, the decision maker will use it directly to make offloading decision. Otherwise, the maker has to get an approximate value by averaging the previous and next value. Moreover, each entry in XML contains two values (i.e. BestVal and CurVal) to record the offloading cost. BestVal is written by the developer at app testing phase while the CurVal is written by the maker at runtime. When a method finishes offloading execution, the maker uses the execution status to update XML file (add new entry or update the CurVal). The CurVal which can well reflect current condition will be used for the following offloading decision. For example, if there exists severely network or server congestion, the CurVal of energy cost will increase dramatically. According to CurVal, the decision maker will schedule the method running locally. Furthermore, to periodically estimate network or server condition, the maker will leverage the BestVal for offloading decision every a period of time. As a result this system can make wiser offloading decision with self-learning.

When a method is decided to be offloaded, the Cloud Handler will carry out a series of procedures as follow:

Step 1: A Message containing application information associated with the social profile is sent to Device Handler. It is useful for loading app and recognizing user in the cloud.

Step 2: The Message is received and parsed to extract application information. Then Device Handler will search Server and App Profiler to find whether this application is supported. If it can be found, the Handler will send 'Found' as acknowledge. Otherwise, it will send 'NotFound' or 'Deny' according to if the social relation of this device is preferred.

Step 3: If 'NotFound' is received by Cloud Handler, it shows that this application is a new brand for the cloud and the cloud is willing to provide the service. Then the whole .apk file will be sent to the Proxy which records app information in Server and App Profile and further sends the app to a designated worker Domain. If 'Deny', the task has to be executed locally.

Step 4: From now on, cloud will support this application. To offload method to the cloud, Cloud Handler sends another Message containing method information such as local method name, real parameters, etc. Then the Proxy chooses the corresponding worker Domain to execute that method. When it is finished, the result will be forwarded back. And the app can continue the following execution in the device.

5.3 Local cloud framework

In order to support large-scale offloading service and adapt to real life, we design a social scheduling based application-isolated model.

As to general service, we deploy several hot apps in the cloud in advance. To avoid the inference between same kinds of app [13], we separate them into different isolated worker Domains. All the apps and worker Domains information are restored in Server and App Profiler. As to task scheduling, different from FIFO or traditional delay priority scheduling scheme, we propose social-based priority scheme. We define the social profile as the composition of visit frequency, residence time and shopping bill in one week as follow:

$$SP = \frac{\#(frequency)}{\#(Expected\ frequency)} + \frac{\#(time)}{\#(Expected\ time)} + \frac{\#(bill)}{\#(Expected\ bill)}$$

The POIs themselves can set the "Expected" values and also can assign different weight to these three parts. If the SP value is over a threshold, the POIs will regard its social profile as preferred.

When the Device Handler receives the Message from device, it will look up the application name in Server and App Profile to choose the right worker Domain. Then according to its social profile, it will use the XEN library-"libvirt" and XenSocket [17] to dynamically assign "bonus" resources to that domain then put the task to the Task Scheduler. The users with stronger social relation (higher SP value) will be scheduled to run quickly, which can fully enjoy those resources. Once the worker Domain gets the task, it loads the app and leverages the Java-reflection to execute the task. After the task finishes, the Device Handler retrieves the corresponding resources back and sends the result back to the device. Taking load balance into consideration, we evenly separate the "hot" applications into different service domains according to the "hot degree" in advance and randomly assign the incoming new applications from users at runtime. With this strategy, we suppose each worker Domain can roughly balance the number of request from users. From the preceding description, our system respectively addresses three key issues in method offloading based cloud computing. Next we will evaluate it with different kinds of application.

VI. Performance Evaluation

We use Dell PowerEdge R410 Sever with XEN version 3.3 to build up virtual cloud on CentOS5 distribution. We create two guest domains with the same configuration [13] as worker Domain. The privilege domain is treated as Proxy. The devices include two Samsung Galaxy S3 smartphones which represent the mainstream configuration and three laptops with Android Virtual Machine. All these machines are linked by 54Mb/s IEEE 802.11g wireless network which is the same network configuration in our Uni-cafe.

6.1 Experiment arrangement and test tools

Generally speaking, the typical applications in current life would be gaming, web surfing, video streaming and chatting. However, all the methods in those applications that should be offloaded can be categorized as CPU-intensive (some calculation in gaming), Memory-intensive (image or video editing) and I/O-intensive methods (resource downloading from web). In this paper, our purpose is to evaluate if our proposed system can well support to those three categories. Therefore, we choose classic Fibonacci series as the CPU-intensive, image process as Memory-intensive and an image downloading as I/O-intensive category.

We utilize the Java API 'currentTimeMillis' to get the execution time. The open-source project PowerTutor, a power monitor Android application, [14] proposes a dedicated power estimation model in terms of CPU, Wi-Fi, GPS, LCD and Cellular. It is able to separately estimate the energy consumption of the preceding five components for every running Android application. We modify some minor incompatible APIs and leverage it to obtain energy consumption of the CPU and Wi-

Fi. All the results are the average data from smartphones and each runs 30 times. Since our system applies to local Wi-Fi environment, we do not take cellular network (3G/4G) into consideration. Meanwhile, we assume the network bandwidth is sufficient local environment and thus we release the bandwidth allocation issue for different consumers in this paper. We just analyze execution time and energy consumption in smartphone and neglect them in the cloud. Meanwhile, in our result the time consumption and energy consumption always share the same trend. So, we only show one of them in the following graphs due to the space limit.

6.2 CPU-Intensive application

Figure 4.1 and 4.2 show the energy cost of running Fibonacci series locally and remotely. We find that offloading can achieve better performance than running locally. That is because local execution energy is cost by CPU and the time complexity of Fibonacci recursive algorithm is $O(2^n)$. Even if calculating the lowest parameter (26), it costs as much as 0.11J. While offloading execution energy is cost by communication and the exchanged data in this app is limited. So the cost remains relatively low to about 0.03J-0.06J. (when offloading 38-44, it need more time to wait for the result from the cloud so the energy cost by Wi-Fi increases) We can draw a conclusion: the app with small input volume and high complexity is greatly adapted to our system. Next, we will evaluate if our system is also suitable for the app with high input volume.

6.3 Memory-intensive application

Image Handler is a typical memory-intensive application which can be further divided into two groups: pure memory-intensive application and composite application. The former does not require much CPU resource while the latter needs both much memory and CPU resource. In our experiments we use Grayscale and Sharpening to represent these two genres.

Figure 5.3 shows the time consumption when running Grayscale and Sharpening. 'Convey' is the amount of time cost by transmitting data packets when offloading. 'Server' is the amount of time cost by running application in cloud. The total time of offloading and execution is the sum of the 'Convey' and 'Server'. The time cost by local execution is denoted 'Phone'. We can see that, although the complexity of Grayscale and Sharpening is respective O(n) and $O(n^4)$, the execution time in the cloud is pretty low and does not show much difference (less than 1s in average). This situation verifies once more our system is suite for high complexity tasks. Since the computing resource in device is critical, the results show dramatically difference. In figure 5. 1-3 the Grayscale execution time in device is less than the data exchanging time such that the energy cost in local is better. While the Sharpening shows the opposite trend and the gap increases with the data size increasing. Therefore we can draw a conclusion: the app with large input volume and high complexity

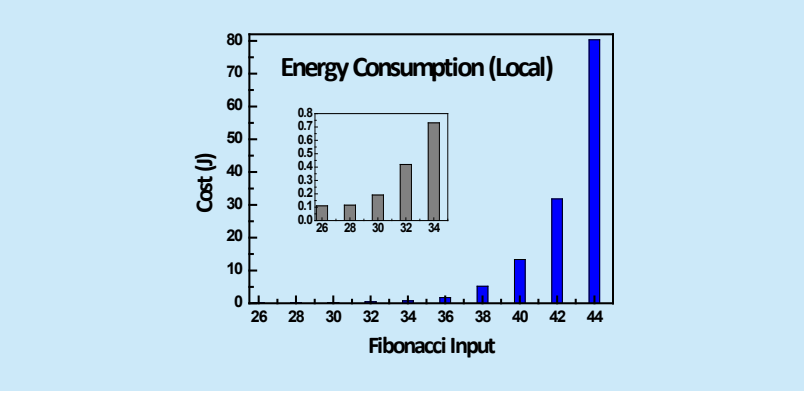

Fig.4.1 *energy cost of local execution*

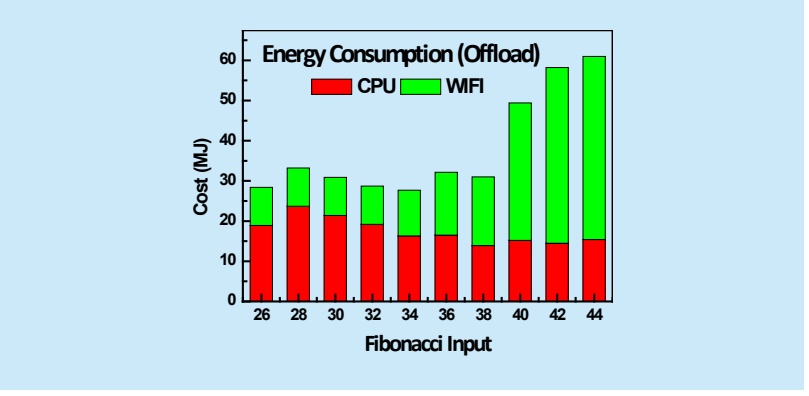

Fig.4.2 *energy cost of offload execution*

is also adapted to our system but pure large input volume tasks is not suitable. So we will dive into the correlation between complexity and input volume for making better offloading decision in future.

6.4 I/O-intensive application

To evaluate the performance of I/O-intensive application, we create a multithread downloading application to obtain picture from RenRen web site. In this experiment we take 'Photo Memory' [15] as an example. Table I shows time and power consumed by local and offloading execution when downloading different amount of pictures.

When the application runs locally, it parses web page, establishes connection, downloads picture for every URL. As for offloading execution, smartphone only need to retrieve picture from the cloud and the rest of all are left with the cloud server. We can see that there is little difference between Offloading and Local execution. This is because the performance of I/O-intensive application is dominated by the network bandwidth wherever the application runs. Therefore, I/O-intensive applications such as chatting, web surfing and etc. in real life do not benefit from the offloading approach.

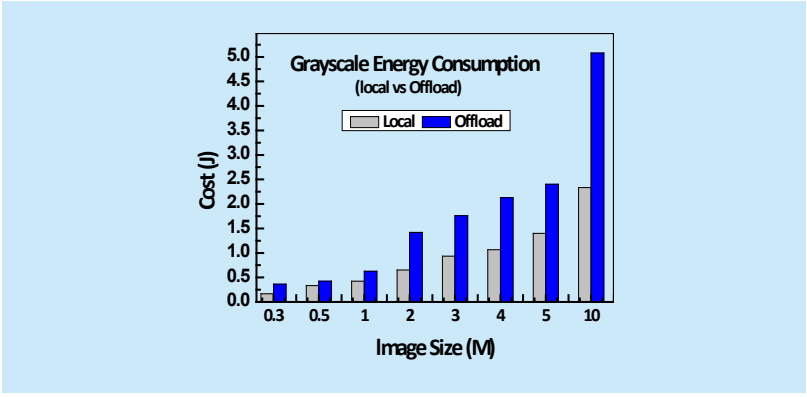

Fig.5.1 *Execution Energy of Grayscale*

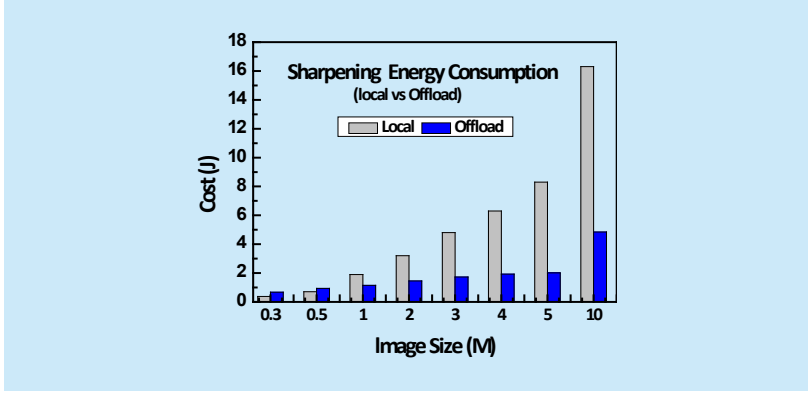

Fig.5.2 *Execution Energy of Sharpening*

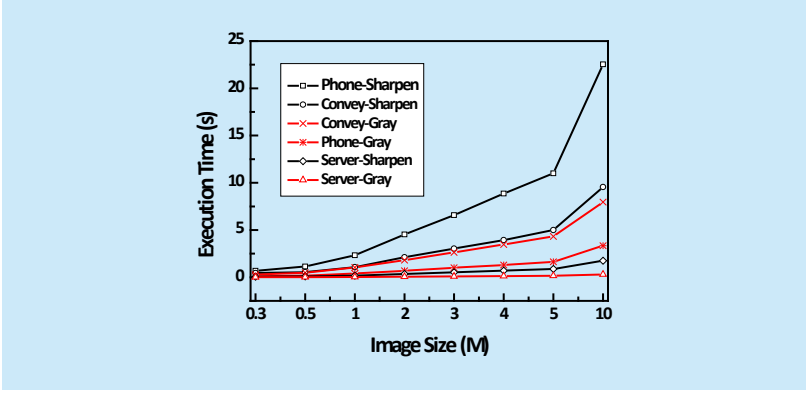

Fig.5.3 *Execution time of Image Handler*

6.5 Concurrent requests analysis

In order to benchmark the performance of concurrent requests, we carry out the following experiments: the two smartphones and three laptops respectively request small volume and high complexity (large volume and high complexity) tasks and one smartphone with high priority, the laptops with the same medium priority and the rest smartphone with low priority.

We aim to find out if there exists greatly inference between tasks with different priority and with same priority. Each device generates 10 requests and we only outline the average execution time.

In figure 6.1 and 6.2 we find that the task with high priority can achieve the performance as well as single execution (less than 2s). This implies our social-based priority scheduling scheme can keep the QoS of old customers. Meanwhile, when it comes to regular customers who share the closeness priority, our system can provide the fair service. (The performance of three medium tasks is roughly the same) In addition, although the performance of low task is worse compared with others, it still outstands that of local execution (20s vs 120s in Fibonacci 44 and 16s vs 22s in Sharp-

ening 10M). In other words, even the customer is the first come to this place, he/she can still benefit from our system. Besides, we also find that in Figure 6.1 if many tasks have occupied some computing resources in one domain, the following task will be badly impacted. (6s vs 20s in Fibonacci 44) This implies that the high complexity app should be separated into different isolated domain. Meanwhile, if many tasks have occupied the bandwidth, the following task will be also badly impacted. (16s is close to local execution: 22s in Sharpening 10M) From this aspect, the app with large volume input should be separated into different isolated domain. Therefore it is better to mix CPU-intensive and Memory-intensive apps rather than set the same kind apps alone. From the preceding result, we can draw a conclusion: our mobile cloud computing system can effectively unleash the heavy parts to the cloud and efficiently support concurrent request with different social priority in real life.

VII. Conclusion

In this paper, we design "Samrt Cafe", a mobile local cloud computing system in cafe (POIs) environment. Our system answers the following key issues: 1) As to application, how to locate resource-intensive components. 2) As to mobile device, how to effectively offload those components to execute in the cloud. 3) As to cloud, how to concurrent support large scale offloading service efficiently. We deploy it in our university Cafeteria. The empirical results show that, by using "Smart Cafe", the CPU-intensive calculation task consumes two orders of magnitude less energy on average; the processing speed of latency-sensitive image sharpening task gets doubled; In addition, the proposed social-based scheduling cloud model can well support concurrent requests from different social profile. We believe, due to its simplicity and practicability, our system can adapt to and be beneficial to more pervasive POIs environment.

Table I *Time and power cost of picture download*

Picture (pages)	Size (M)	Offloading		Local	
		Time(s)	Power(J)	Time(s)	Power(J)
10	3.2	3.63	3.3	4.71	4.9
50	8	12.49	12	13.76	14.2
100	13.4	22.42	20.9	25.76	24.4
150	20	31.88	29.4	36.81	35.9
200	28.4	74.85	50.8	82.9	59.5

Acknowledgements

This work was supported by the Research Fund for the Doctoral Program of Higher Education of China (No.20110031110026 and No.20120031110035), the National Natural Science Foundation of China (No. 61103214), and the Key Project in Tianjin Science & Technology Pillar Program (No.13ZCZDGX01098).

References

[1] Hoang T. Dinh, Chonho Lee, Dusit Niyato, and

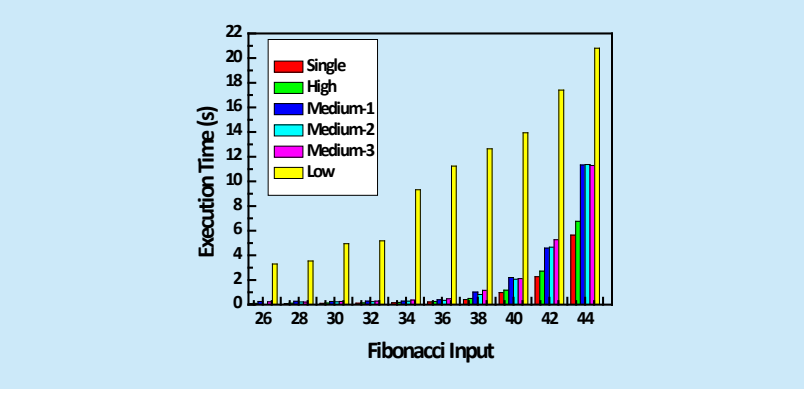

Fig.6.1 *Execution time of concurrent small volume and high complexity task*

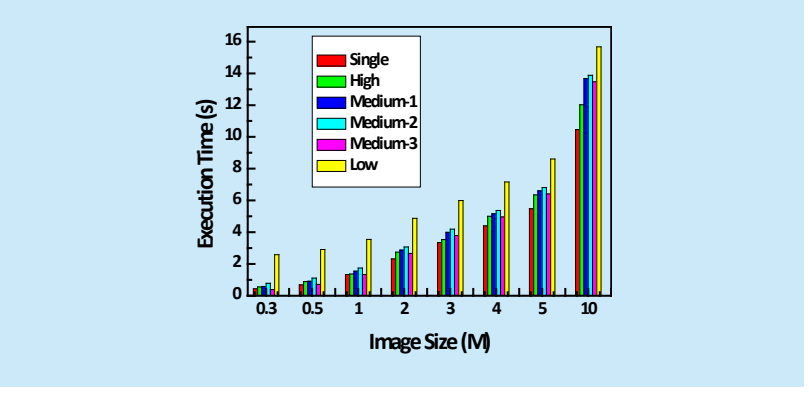

Fig.6.2 *Execution time of concurrent large volume and high complexity task*

Ping Wang. "A Survey of Mobile Cloud Computing: Architecture, Applications, and Approaches." Wireless Communications and Mobile Computing 2011.

[2] Mahadev Satyanarayanan, Paramvir Bahl, Ramon Caceres, and Nigel Davies. "The case for Vm-based cloudlets in mobile computing" Pervasive Computing, IEEE, 2009.

[3] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, et al. "Clonecloud: Elastic execution between mobile device and cloud." In Proceedings of the 6th European Conference on Computer Systems, April 2011.

[4] M. Papandreaz, M. Zignaniy, *et al*. "How many places do you visit a day" in Pervasive Computing and Communications Workshops (Percom), 2013

[5] S. Gordon, Jamshidi, S. Mahlke, *et al*. "COMET: Code Offload by Migrating Execution Transparently" in Proceedings of OSDI 2012.

[6] A. Pathak, *et al.* "Enabling Automatic Offloading of Resource-Intensive Smartphone Applications" Technical Report, Purdue University 2011.

[7] E. Cuervo, *et al*. "MAUI: making smartphones last longer with code offload." In Proceedings of the 8th international conference on Mobile systems, applications, and services. ACM, 2010.

[8] S. Kosta, A. Aucinas, *et al*. "ThinkAir: Dynamic resource allocation and parallel execution in cloud for mobile code offloading", In IEEE InfoCom 2012.

[9] Marco V. Barbera, Sokol Kosta, Alessandro Mei, and Julinda Stefa. "To Offload or Not to Offload? The Bandwidth and Energy Costs of Mobile Cloud Computing" in IEEE InfoCom 2013.

[10] G. Chen, B. Kang, M. Kandemir, et al., Studying energy trade-offs in offloading computation/compilation in java-enabled mobile devices, IEEE Transactions on Parallel and Distributed Systems, 2004.

[11] Shumao Ou, Kun Yang and Jie Zhang., "An effective offloading middleware for pervasive services on moblie devices", Pervasive and Mobile Computing Volume 3, Issue 4, August 2007, Pages 362–385.

[12] A. Saarinen, M. Siekkinen, Yu Xiao, *et al*. "SmartDiet: Offloading Popular Apps to Save Energy" in ACM SIGCOMM 2012.

[13] Lingjun Pu and Jingdong Xu. "Measurements Study on the I/O Performance of Virtualized Cloud System," in IEEE International Conference on Networking, Architecture, and Storage (NAS) 2012.

[14] Lide Zhang, Birjodh Tiwana, Zhiyun Qian, Zhaoguang Wang, *et al*. "Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones." In Proceedings of CODES, 2010.

[15] http://zhan.renren.com/ournankai

[16] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt and A. Warfield, "Xen and the art of virtualization," in ACM SOSP 2003.

[17] Xiaolan Zhang, Suzanne McIntosh, Pankaj Rohatgi, and John Linwood Griffin, XenSocket: A High-Throughput Inter-domain Transport for Virtual Machines. 2007

Biographies

PU Lingjun, is currently a Ph.D. candidate at College of Computer and Control Engineering, Nankai University, China. His research focuses on mobile cloud computing, wireless networking and opportunistic routing. Email: pulingjun@mail.nankai.edu.cn

XU Jingdong, is currently a Professor in College of Computer and Control Engineering of Nankai University and leading the Computer Networks (NET) Research Group. Her main research interest is sensor, vehicle ad-hoc network, network security and management, and opportunistic network and computing. Email: xujd@nankai.edu.cn

YU Bowen, is a Ph.D. Student in College of Computer and Control Engineering of Nankai University, China. His current research is in traffic classification and network security. Email: yubowen@mail.nankai.edu.cn

ZHANG Jianzhong, is a Professor in College of Computer and Control Engineering of Nankai University, China. He is interested in computer networks, network security and mobile computing. Email: zhangjz@nankai.edu.cn