# An Investigation of Different Computing Sources for Mobile Application Outsourcing on the Road

Mohammed Anowarul Hassan and Songqing Chen

Dept. of Computer Science
George Mason University
mhassanb@gmu.edu,sqchen@cs.gmu.edu

**Abstract.** Mobile applications are growing fast due to pervasive usage of mobile devices. With inherently limited on-device resources, plenty of research has been conducted on job partitioning/outsourcing strategies to execute mobile computing tasks on external sources, such as public clouds or nearby computers. However, little is known about the performance difference to mobile users on these external computing sources.
In this paper, considering the user's response time and the battery power consumption on mobile devices, we first show that outsourcing mobile applications to public clouds may not outperform outsourcing to nearby residential computers, particularly for delay sensitive applications. To facilitate efficient mobile outsourcing to residential computers, we propose to build a framework RoseMic (ROad-SidE-MobIle-Computing). In RoseMic, a resource overlay network is built with users' idle residential (home) computers. To encourage the sharing of idle residential computers, RoseMic also includes a credit based incentive mechanism that can be enforced automatically without users' interferences in order to defeat collusion attacks. To demonstrate the performance of RoseMic, we run several real-world applications. The results show that RoseMic outperforms Amazon EC2 by 3 times and 4 times on average in terms of response time and the battery power consumption, respectively.

## 1 Introduction

With the fast development of micro-chip and wireless technologies, mobile devices are gaining increasing popularity. According to International Data Corporation, the total number of hand-held devices sold in 2009 is 1,127.8 million [4]. Among them, 174.2 millions (15.5%) are smartphones, which is a 15.1% increase from the previous year.

The pervasive usage of mobile devices has enabled fast growth of mobile applications. Compared to traditional mobile phones that are mainly used for voice communications, today a smartphone is capable of common tasks that were only possible on desktop computers, such as surfing the Internet, taking and editing pictures, gaming, document processing, etc.

However, mobile devices, albeit their fastly increasing CPU speed and memory size, are not as capable as modern desktop computers when running these applications. In particular, mobile devices are ultimately constrained by the limited battery supply and a prolonged computation process or a computing intensive application can quickly exhaust the limited battery power.

To address the fundamental challenge posed by the limited on-device resources for computing-intensive tasks, plenty of research has been conducted on designing various job partitioning/outsourcing strategies [7, 12, 15] to outsource mobile computing tasks to external sources. Today external computing sources are widely available, such as public clouds or nearby surrogate computers. However, little has been done to investigate the outsourcing performance to mobile users on different computing sources. This is particularly important for some delay sensitive mobile applications. In practice, a mobile user in motion may only have sporadic WiFi connections with very short connection duration. For example, previous work [14] has shown that with a typical WLAN with 200 meters of range, a user in motion with a speed up to 120 Km/h can have Internet access for about only 6 - 12 seconds. It is thus very desirable for a mobile user to complete the outsourcing and get the result back as soon as possible, best in this time frame.

In this paper, we set to investigate where to effectively outsource mobile applications on the road. Considering the response time to the user and the battery power consumption on mobile devices that are critical to mobile users in motion, we first show that outsourcing mobile applications to clouds may not outperform outsourcing to nearby residential computers, particular for delay sensitive applications. Thus, to facilitate efficient mobile outsourcing to roadside residential computers, we propose to build a framework RoseMic (ROad-SidE-MobIle-Computing), aiming to reduce the user's response time and battery power consumption. In RoseMic, a resource overlay network is constructed with participating users' idle residential (home) computers. Furthermore, to encourage users to participate in this resource overlay, an incentive mechanism is built and enforced automatically without any user's interference in order to defeat collusion attacks.

To demonstrate the performance of RoseMic, we have built a prototype and experimented RoseMic with several real-world applications, including text searching, face detection, and image processing. The results show that RoseMic not only outperforms on-device computing by 8 times and 10 times in terms of response time and the battery power consumption, respectively, but also outperforms Amazon EC2 by 3 times and 4 times on average, respectively.

The remainder of the paper is organized as follows. We present our motivation in Section 2 and RoseMic overview in Section 3, followed by the detailed design in Section 4. We present some preliminary evaluation results in Section 5 and discuss related work in Section 6. We make concluding remarks in Section 7.

## 2 Motivation

Previous research has proposed that mobile devices can outsource computing intensive jobs to public clouds or surrogate computers. To test the performance of outsourcing to the public clouds and nearby residential computers, we have conducted some preliminary experiments with Amazon EC2 and our local computers. The experiment is to find a string in a text file. Three approaches have been tested out.

- **Android**: This is to perform computation on the mobile device itself. In the experiment, we use Google Nexus one SmartPhone with Android 2.2 Operating System and 1 GHz CPU and 512 MB RAM.
- **Amazon EC2**: This is to send the data file and the computing task to Amazon EC2. We rent EC2 and follow the instructions to get our rented portion initialized first. Our portion of slice uses a CPU of 5 GHz and 1.7 GB of memory. The wireless network speed is 300 Kbps on average.
- **Residential Computers**: This is to send the data file and the computing task to nearby computers. In the experiment, we use the Android phone to send to the local computer in the same network via 802.11b with a speed of 10 Mbps. The local computer has a CPU of 2 GHz and 3.2 GB of RAM.

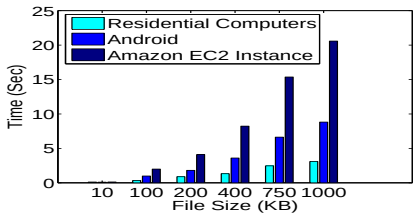



**Fig. 1.** Response Time

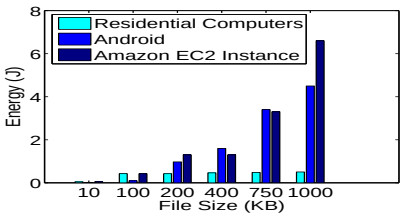



**Fig. 2.** Energy Consumption

Figure 1 shows the user response time when the same program is executed with different approaches along the increase of the file size. It also shows that although EC2 has a much faster CPU, the local machine outperforms EC2 by about 7 times and the performance on EC2 is even worse than the performance when it is executed on the mobile device itself. Figure 2 further shows the corresponding energy consumed on the mobile device for executing the program with different approaches. Not surprisingly, outsourcing to nearby computers consumes the least amount of energy on Android based on Power Tutor [5].

We have also conducted experiments with several other applications and obtained similar results. These results show that when outsourcing mobile computing tasks, the network tranferring time, i.e., the available bandwidth, has to be taken into consideration and sometimes this may be a dominant factor. Under such situations, outsourcing to nearby residential computers may be more beneficial than to public clouds.

## 3 Overview of ROad-SidE-MobIle-Computing (RoseMic)

RoseMic is proposed to assist mobile computation outsourcing by leveraging nearby residential computers. In RoseMic, we assume that data transferring in this procedure can only be done through WiFi instead of cellular connections considering its cost and relatively small bandwidth.

Figure 3 depicts the architecture of RoseMic. As shown in this figure, there is a resource overlay that a participating mobile user can leverage for computing

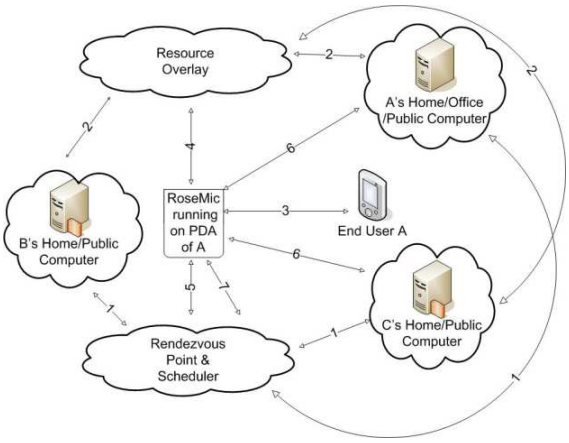



**Fig. 3.** Architecture of RoseMic

in motion. The resource overlay consists of normal users' residential (home) computers. In general, when a user is in motion, her home computer is often idle, offering available CPU cycles and WiFi connections that can be utilized. A user can register her home computer in the resource overlay (**Step 2**) before her departure through the rendezvous points (**Step 1**). At present, we assume there is one to one map between the mobile user and her home computer on the overlay. It is easy to expand to the one to many case.

When the user is on the road and wants to execute some computing intensive application, she submits the job to the RoseMic client running on her mobile device (**Step 3**). The RoseMic client will contact the overlay (**Step 4**) by visiting a well-known bootstrapping site. Such contact could be done through the cellular connection if WiFi is not available. In the response to this request, the user is directed to the rendezvous point (**Step 5**). Based on the geographical location of the mobile user, a list of nearby residential computers on the overlay are identified by the rendezvous point. The rendezvous point also maintains the credit information for each user. The scheduler running on the rendezvous point further selects some of these computers based on its scheduling policy. The scheduler can take a lot of factors into consideration, such as the current load on the machines, the distance to the user, the available credit of the requesting machine, etc.

Once the list of residential computers in the connectivity range is determined, the list of the machines is sent back by the rendezvous point to the mobile user (**Step 5**). From now on, all communications are done through WiFi connections. Based on the list, the mobile user connects to these nearby computers and can start the execution. If the RoseMic scheduler decides to execute the job on-device, then it does so and returns the result to the user (**Step 3**). If outsourcing is more economical, RoseMic then submits the job to the most appropriate residential computer and waits for the result (**Step 6**). The RoseMic then also returns the result to the user's application (**Step 3**). Upon the completion of the task, RoseMic automatically sends the completion information to the rendezvous point

so that the credit scores of the involved users are updated (**Step 7**). That is, the credit of the mobile user who requests the computation outsourcing should be deducted accordingly and the credits of owners of the computing machines should be increased.

Note that in practice RoseMic does not prevent the user from leveraging public clouds when necessary. For example, when there is no nearby residential computers available, public cloud can be used.

## 4 Design of RoseMic

The previous section shows that RoseMic consists of (1) a resource overlay, (2) an incentive model and (3) a scheduler. In this section, we further present our design of these three parts.

### 4.1 Resource Overlay Network

To utilize nearby residential computers, in RoseMic, a resource overlay network is constructed by mobile users' residential computers that would otherwise be idle. Note that our assumption is that there is a one-to-one map between a mobile user and her residential computer on the overlay. That is, there is a same `id` used to identify the mobile device and the corresponding overlay computer. In addition, a rendezvous point is used to store the credit made by the corresponding overlay computer for each mobile user. Basically, the rendezvous point provides directory services and maintain a database of record `<id, location, status, credit>` for each user. Note the location is the geographical location of the mobile user's residential machine, and the status is the load state on that machine. The location is denoted as a pair of `<latitude, longitude>`. The status could be one of the following: `idle, medium, busy`, and is determined simply based on the CPU utilization on the machine. In the following context, we assume the node and the residential computer are interchangeable unless noted otherwise.

**Node Joining** If a user wants to register her node to offer services, she can instruct the joining node to first contact the bootstrapping site via a Web browser (assume the same Web site www.RoseMic.org). In the response, the rendezvous point (RP) is returned to the joining node. Subsequently, the joining node contacts the RP for joining. In the joining request to the RP, the `id` and `location` information is also required. However, we do not assume GPS or GPS-capable functions on the home computer. Instead, we rely on the Google Geolocation service by enabling the "sharing your location" in the web browser. Such a request leverages Google Geolocation service to identify the location of the requesting computer based on nearby access points and other wireless information [3].

With the `id` and `location` information, the RP can retrieve its available credit from the database. Furthermore, the RP sets this computer status to idle initially, and sorts the available computers based on their location. After joining the resource overlay network, the joining node does not keep its connection with the RP. Instead, a participating node sends periodic heartbeat message to the RP to notify it aliveness.

**Node Departure** A user's residential computer may depart from the resource overlay network for many reasons. For example, the user may want to use her computer exclusively or it may simply crash. Accordingly, we classify node departure into two categories. The first is normal departure. In a normal departure, the node notifies the RP about its un-availability and requests to be taken off from the available list maintained by the RP. Correspondingly, the RP removes this node from the available computer list. Since a participating node periodically sends keepalive message to the RP, it is easy to deal with abrupt departure. Basically, if there is no keepalive message received for a certain timeout period, the node is assumed to be dead. Correspondingly, the node is removed from the available list.

### 4.2 Credit-based Incentive Model

To encourage the contribution of idle CPU cycles and bandwidth from residential computers, we propose to build a credit-based incentive mechanism on the overlay. The motivation behind users' offering their residential computers is as follows. While a user is in motion, her residential computers is often idle. So is the residential WiFi connection. Thus, a user in motion can use other users' computers for computation as a return of offering her computer and WiFi connection for others when she is on the road. To guarantee fair sharing, users who offer CPU cycles from their computers get credits that they could use in the future (when they need to use other users' computers or WiFi connections).

To keep track of the credit of each user, the credit is stored on the RP. To deal with collusion attacks, RoseMic also includes a credit system. While more details will be provided later, the high level idea is: upon the completion of a task, a transaction module in RoseMic automatically reports to RP about the computing tasks conducted on the relevant computers. Basically, the transaction module reports 1) the `id` of computing requester, 2) the `id`s of the computers used, 3) the amount of CPU cycles and bandwidth being used from each of the involved computers. The RP uses this information to calculate the credits that should be deducted from the requester and distribute these credits to the involved `id`s proportionally.

For example, if user A uses the residential computer of C, the credit of user C should be increased by the equation 1:

$$credit_C^A += \alpha \times C_A + \beta \times D_A, \tag{1}$$

where $C_A$ represents the total CPU cycles used by user A, $D_A$ represents the total bandwidth usage by user A, denoted by the amount of data transferred to the computer of C from A's mobile device for the computation. Based on the same principle, user A's credit is reduced. Note in equation 1, $\alpha$ and $\beta$ are normalization factors that can be determined experimentally and/or based on the resource demand and supply. Initially, each user has no credit but we allow negative credits in the system.

Our credit-based incentive model is enforced without direct involvement of the users. This is to deal with collusion attacks that some users may launch to claim credits for bogus transactions. Unless our transaction module in RoseMic

is attacked and modified, no collusion attacks could be launched. In addition, we can also enforce that a user cannot use other users' computer unless her computer is on the resource overlay in order to deal with free-riders.

### 4.3 RoseMic Scheduler

The RoseMic scheduler is responsible for finding the most appropriate residential computers for the mobile user who requests the computation outsourcing service. In our current design, the scheduler runs on the RP.

In finding the most appropriate computers to execute the task, the RoseMic scheduler considers the following:

1. User's credit: A user with a high credit score has a high priority to get her job outsourced. This is to encourage more users to participate in the resource overlay network.
2. Job Type: When requesting the computation outsourcing, the mobile user also indicates the job type. Based on both the CPU and the bandwidth demand and the data locations, nearby residential computers or the user's home computers are determined.
3. Bandwidth: In searching for the most appropriate residential computers for outsourced computation, the computer with the highest bandwidth is always selected first if other conditions are the same.
4. Node Status: Since RoseMic targets delay-sensitive applications, the status of selected computers may affect the response time. Furthermore, it is also important for the RP to maintain load balance across different residential computers on the resource overlay.

Considering the above factors, RP employs its optimization function by comparing the predicted response time of different task assignment strategies. It takes a number of input and outputs the best execution strategy.

**Algorithm 1** RoseMic Scheduler(I,J,L)

1: $LC \leftarrow$ The set of available computers nearby including the mobile device and home computer of the user calculated from I,J and L
2: $LCF \leftarrow$ empty set
3: **for** each node $i$ in $LC$ **do**
4: **if** $J_c \leq i_c$ **then**
5: add $i$ to $LCF$
6: **end if**
7: **end for**
8: **for** each node $i$ in $LCF$ **do**
9: $i_{dt} \leftarrow J_D - i_{ds}$
10: $candidate\_value_i \leftarrow (i_c - J_C) \times \gamma - (i_{dt} \div i_{bw}) \times \sigma - (|L - i_l| \div i_{bw}) \times \lambda$
11: **end for**
12: **return** node i = max{$\forall$ i $\in$ LCF: $candidate\_value_i$}

In the above algorithm, `I` represents the user's `id`, from which the credit of the user can be retrieved. If multiple users compete for the same residential

computers, user with higher credit gets the priority. `L` is the `location`, which is needed to find the nearby computers available for computation outsourcing. `J` represents the job type. $J_C$ indicates the CPU cycle demanded, and $J_D$ indicates the amount of data needed to the transfered for computation. `LC` is the list of the available computers, which holds the following informations for each node in that list: 1) $i_c$: computation power available on node $i$; 2) $i_l$: location of node $i$; 3) $i_{bw}$: bandwidth of the channel from the mobile device to the node $i$; 4) $i_{ds}$: amount of data stored in prior for a job J in the node $i$; 5) $i_{dt}$: amount of data needed to be transferred from the mobile device to node $i$. $\gamma$, $\sigma$ and $\lambda$ are constants to add weight on different components. Note that we assume profiling can be used to obtain these parameters for mobile applications.

Note that we include the mobile device and the home computer of the user as potential candidates to execute computation as well, because computation may be economical to execute entirely on the mobile device or the user's home computer, where most of the data may reside.

As shown in the algorithm, in the first step, RoseMic reduces the set of available computers nearby $LC$ to the set of feasible computers $LCF$. If any node has enough CPU to execute the job, it is a candidate and added to set $LCF$. The candidate value for each node $i$ is calculated based on the latency, bandwidth, amount of data needed to be transferred and computation power available on that node. The scheduler then selects the node with highest candidate value to outsource the computation.

## 5 Preliminary Evaluation

To evaluate the performance of RoseMic, we conduct experiments with a RoseMic prototype and compare its performance against when executing on the mobile device as well as executing on the public cloud Amazon EC2. In these experiments, the credit system is not being evaluated as these experiments are conducted locally as proof-of-concept.

### 5.1 Experiment Setup

We use Google Android Nexus One with 1 GHz CPU and 512 RAM as the mobile device for the local execution. We use computers with dual-core CPU with 2GHz and 2 GB RAM to emulate overlay residential computers. The remote EC2 Ubuntu instance has a 5 GHz CPU with 1.7 GB of RAM. We use Power Tutor [5] to measure the power consumed by the applications running on the smartphone. The WiFi is 10 Mbps and the average bandwidth from the Android Phone to EC2 instance is around 300 Kpbs on average.

We have conducted experiments with three applications, 1) Text Searching, 2) Face Detection, and 3) Image Sub-Pattern Searching.

In the experiments, we emulate a 3 user model and they have identical residential computers and mobile devices. The latitude and longitude of the first user's residential computer is 38.901222 and -77.26526, while the other two users' residential computers are at 38.846224 (Lat.) and -77.306373 (Lon.). The EC2

instances rented are at Northern Virginia Data Center of Amazon. We profile each application to deduce the average CPU cycle and data transfer requirement and we run each application in the following different environments.

- **On-device*(OD)***: This is to run the application on the mobile device directly.
- **Computation+Data*(CD)***: This is to outsource the computation program and the data. The RoseMic client running on the mobile device gets connected to the resource overlay and the rendezvous point to explore the neighboring residential computers and outsource the computation with the data file.
- **Computation+Data+Node Failure*(CD+F)***: This is to consider the node failure in the above environment to study the impact of node failure. When a node fails, we contact the resource overlay and the rendezvous point once again to find another neighboring residential computer and restart the job. To emulate node failure case, we deliberately turn off one computer in the middle of an on-going computation when the computation is 90% completed. Then RoseMic detects the failure based on timeout and it contacts resource overlay and rendezvous point to find another nearby residential computer.
- **Computation*(C)***: This is to outsource only the computation for these applications. This is to emulate the scenario when the selected residential computer is the user's home computer, which has the data of the task and the mobile device only needs to transmit a small portion of data.
- **Computation+Node Failure*(C+F)***: This is to consider the node failure in the above environment. Note that here for both the failed node and the new node, we outsource only the computation. We emulate the node failure case as we have done for CD+F.
- **EC2 *(EC2)***: We outsource the computation to the remote amazon EC2 instance with 5 GHz of CPU and 1.7 GB of memory. We assume that EC2 is always available and 100% reliable.

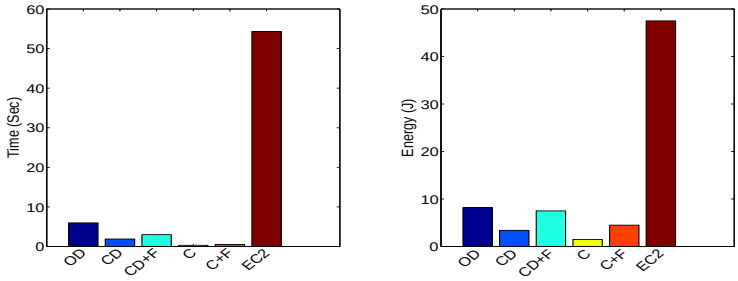
We use Java and outsource the computation class in byte code format and use java dynamic class loader to execute the computation. We assume the computing environment is set in advance. In these experiments, we mainly focus on the response time and the energy consumption on the mobile device.

### 5.2 Experiment Results

In this section, we describe the performance of the different approaches we have tested for the three applications. We repeat each experiment five times and present the average of the results.

**Text Search** In this experiment, the user searches a string in a text file and the frequency of occurrence of that string is returned to the user. This simple string counting application takes an input file of 2.6 MB. We use string matching to find the total number of occurrence of that string in that text file.

Figure 4 shows the performance of the application when it is executed on the Android and the computation is outsourced. Figure 4(a) shows the response time of the execution. Figure 4(b) shows the corresponding energy consumption on the mobile device. In this application, if the data needs to be outsourced, it is




(a) Response Time (Sec) (b) Energy Consumption (J)

**Fig. 4.** Text Search

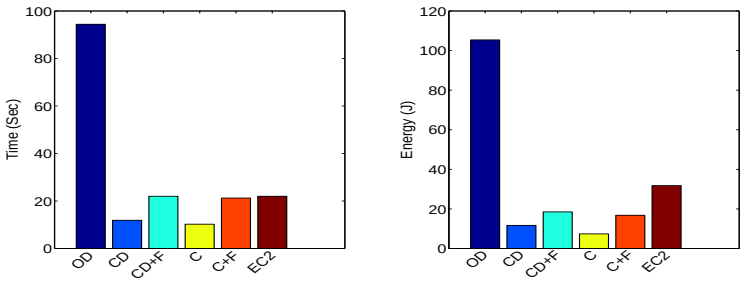
2.6 MB. Otherwise, only the computing program needs to be outsourced, which is 1 KB.

As shown in Figure 4, outsourcing to EC2 results in the worst performance in terms of both the response time to the user and the amount of energy consumed, which are significantly larger than if the application is executed on the mobile device itself. On the EC2, the response time is over 50 seconds. If we consider the average connection time of a mobile user with a roadside WiFi ranges between 6-12 seconds [14], this is impossible for a mobile user to get the result in time in the same communication session although EC2 has a faster CPU speed (note that Amazon Northern Virginia Center is in the same region where these experiments are conducted). This would be a critical problem for delay sensitive mobile applications that a user waits to get the result back.

Figure 4 also shows that although outsourcing to nearby residential computers demands some bandwidth for file transferring, the response time and the total energy consumption on the mobile device are 69% and 59% less than that when the application is executed on the mobile device itself, respectively, indicating the benefit of the outsourcing.

As residential computers may not be reliable for many reasons (e.g., a user comes home and wishes to use her computer dedicatedly), we also study the node failure for this application. Figure 4 shows with node failure, the performance of outsourcing still outperforms the on-device computing in terms of both the response time and the total energy consumed on the mobile device, although there is a 76% and 200% increase compared to if there is no node failure.

**Face Detection** In this experiment, we take a picture of a human face and try to match it with all the pictures in a folder previously taken. We use Cross-Correlation Function [2] to find the correlation between an image pair. Based upon that, we detect a particular person. We have each image in a different jpg file. The correlation between the files has been calculated by taking input from three different streams for 3 RGB values. The resultant size for the reference images is 575 KB in total and the newly taken image size is 145 KB. So the total size of the data file is 720 KB, and the computation program is 3 KB.




(a) Response Time (Sec) (b) Energy (J)

**Fig. 5.** Face Detection

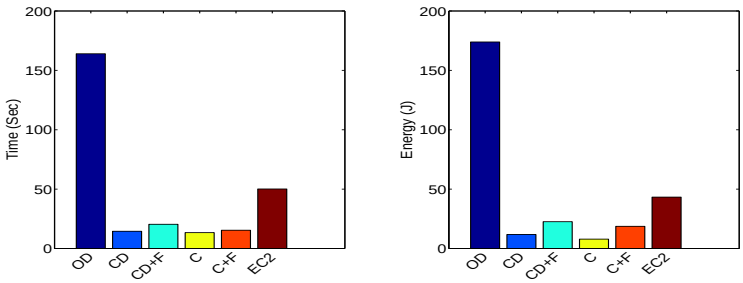
Figure 5 shows the performance results when the program is executed in different environments. For this application, Figure 5(a) shows that executing on the Android takes the longest time of about 94.5 seconds. In all the outsourcing scenarios, the response time is significantly reduced. Not surprisingly, the corresponding energy consumption is the largest for the on-device execution.

While in all the outsourced computation scenarios, both the response time and the energy consumption are reduced, the reduction when the program is outsourced to the nearby residential computers is more pronounced than when the program is outsourced to EC2: on the residential computer, the response time is about 10.25 seconds and 11.90 seconds without or with the data transferred. This indicates that it is possible for the user to get the result back with the same connection when the user is in motion. Correspondingly, the energy consumed is only about 23% and 36%, respectively, when the computation is outsourced to nearby residential computers.

Even when there is node failure, Figure 5 shows that both the response time and the energy consumption increases by 107% and 127% compared to their counterpart without any node failure, the results are still comparable or better than those when outsourcing to EC2, in terms of response time and energy consumption, respectively.

**Image Pattern Search** In this experiment, we take a picture and try to find the picture as a part of another large picture in a folder previously taken. We use Cross-Correlation Function [2] and 2D Logarithmic Search [13] to find the sub-image. We have each image in a different jpg file. The correlation between the files has been calculated by taking input from three different streams for 3 RGB values. The resultant size for the reference image is 1.7 MB and the newly taken image size is 260 KB.

Figure 6 shows the performance results when the program is executed in different environments. For this application, Figure 6(a) shows that executing on the Android takes the longest time of about 163.9 seconds. In all the outsourcing scenarios, the response time is significantly reduced. Correspondingly, the energy consumption is the largest for the on-device execution.




(a) Response Time (Sec) (b) Energy (J)

**Fig. 6.** Image Pattern Search

While in all the outsourced computation scenarios, both the response time and the energy consumption are reduced, the reduction when the program is outsourced to the nearby residential computers is more pronounced than when the program is outsourced to EC2: on the residential computer, the response time is about 13.37 seconds and 14.52 seconds without or with the data transferred. Correspondingly, the energy consumed is only about 10% and 11%, respectively, when the computation is outsourced to nearby residential computers without or with the data transferred respectively.

When there is node failure, Figure 6 shows that both the response time and the energy consumption increases by 40% and 136% compared to their counterpart without any node failure. These results are still much better than those when outsourcing to EC2.

*Summary:* The experimental results show that while computation is outsourced to local residential computers, the overall performance is better than when the computation is outsourced to EC2, though EC2 is much more powerful than nearby residential computers in terms of the CPU speed. The average gain for the response time is about 8 times and 3 times compared to if the computation is executed on Android or outsourced to EC2. The corresponding energy reduction is 10 times and 4 times, respectively.

We also note that EC2 performs the worse than on-device execution in *Text Searching* but performs better in the other two applications. This is due to the nature of the applications. *Face Recognition* and *Image Pattern Searching* applications are more CPU intensive than *Text Searching*. Thus, even executing on Android is faster than outsourcing to EC2, where data transferring becomes a dominant factor.

Note that the above results are case studies because a different user who has a different distance to EC2 or have conducted experiments on EC2 at different times or have conducted experiments with different applications with different data size and bandwidth may have different results.

## 6 Related Work

Outsourcing computing tasks from mobile devices to powerful computing resources, also referred to as remote execution, has been researched for over a decade [7, 12, 15, 8]. The increasingly popular and pervasive usage of mobile devices makes this imperative. To outsource the computation, two classes of approaches have been studied. The first is to create the computing environment without modifying the applications. For example, work [9] proposes a full virtual machine clone technique that can enable the applications to be run without any modification on the cloud. This approach, however, has to consider the significant overhead of cloning the mobile environment, which may be expensive and delay the computing process, whether on the public computing clouds or nearby surrogate computers [16].

Instead of cloning the computing environment, the second approach is to partition the job between the mobile device and external computing resources [7, 8, 10]. Many of existing studies mainly aim to simplify such a process. Balan et al. [7, 8] propose to augment the computation and storage capabilities of mobile devices by exploiting the nearby (surrogate) computers. Rudenko et al. [15] suggests that if the total energy cost of sending the task else where and receiving the result back is lower than the cost of running it locally, then remote process execution can save battery power. Flinn et al. [12] also propose a similar idea, in which remote execution simultaneously leverages the mobility of mobile devices and the richer resources of large devices. Studies [17, 10] demonstrate the ability to partition the application and associate classes and thus outsourcing them. MAUI [11] is recently proposed to partition the program dynamically and submit it on surrogate computers.

Nevertheless, there is little research on where mobile applications should be outsourced. We consider this problem because the response time and the battery power consumption are critical to mobile users in motion. In addition, RoseMic leverages the idle CPU cycle and bandwidth sharing to assist mobile application outsourcing on the road. This is different from SETI@home [6] and BONIC [1], where participants purely voluntarily contribute their CPU cycles to a scientific problem. In RoseMic, participants share CPU and bandwidth with each other in a P2P fashion in order to get services.

## 7 Conclusion

The pervasive usage of mobile devices demands a flexible and effective computation outsourcing mechanism. While a lot of work has been conducted on job partitioning/outsourcing, in this study, we have investigated the effectiveness of outsourcing to nearby residential computers, particularly for delay-sensitive applications. To reduce the user response time and the energy consumption on mobile devices, we have designed a framework RoseMic to effectively support mobile computation outsourcing. We have experimented with several applications and our preliminary results show that our approach can more effectively improve the user response time and reduce the energy consumption on mobile devices than outsourcing to public clouds.

## 8 Acknowledgement

We appreciate constructive comments from anonymous referees. The work is partially supported by US NSF under grant CNS-0746649 and and AFOSR under grant FA9550-09-1-0071.

## References

1. Boinc. http://boinc.berkeley.edu/.
2. Cross Correlation . http://en.wikipedia.org/wiki/Cross-correlation.
3. Google Geolocation Service. http://code.google.com/p/gears/wiki/GeolocationAPI.
4. International Data Corporation : Press Release 28 Jan and 4 Feb, 2010. http://www.idc.com/.
5. Power Tutor. http://ziyang.eecs.umich.edu/projects/powertutor/index.html.
6. Seti Home. http://setiathome.berkeley.edu/.
7. Rajesh Balan, Jason Flinn, M. Satyanarayanan, Shafeeq Sinnamohideen, and Hen-I Yang. The case of cyber foraging. In *Proceedings of the 10th ACM SIGOPS European Workshop*, Saint-Emilion, France, July 2002.
8. Rajesh Krishna Balan, Darren Gergle, Mahadev Satyanarayanan, and James Herbsleb. Simplifying cyber foraging for mobile devices. In *Proceedings of The 5th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, San Juan, Puerto Rico, June 2007.
9. Byung Gon Chun and Petros Maniatis. Augmented smartphone applications through clone cloud execution. In *Proceedings of the 12th Workshop on Hot Topics in Operating Systems (HotOS)*, Monte Verit, Switzerland, May 2009.
10. Byung Gon Chun and Petros Maniatis. Dynamically partitioning applications between weak devices and clouds. In *Proceedings of ACM Workshop on Mobile Cloud Computing & Services (MCS)*, San Francisco, CA, USA, June 2010.
11. Eduardo Cuervo, Aruna Balasubramanian, Dae ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: Making smartphones last longer with code offload. In *Proceedings of The 8th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, San Francisco, CA, USA, June 2010.
12. Jason Flinn, Dushyanth Narayanan, and M. Satyanarayanan. Self-tuned re-mote execution for pervasive computing. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS)*, Schloss Elmau, Germany, May 2001.
13. J. R. Jain and A. K. Jain. Displacement measurement and its application in interframe image coding. In *IEEE Transactions on Communications*, volume 29, December 1981.
14. Jőrg Ott and Dirk Kutscher. Drive-thru internet: Ieee 802.11b for automobile users. In *Proceedings of IEEE InfoCom*, Hong Kong, March 2004.
15. Alexey Rudenko, Peter Reiher, Gerald J. Popek, and Geoffrey H. Kuenning. Saving portable computer battery power through remote process execution. In *Proceedings of Mobile Computing and Communication Review (MC2R)*, 1998.
16. Mahadev Satyanarayanan, Paramvir Bahl, Ramon Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. In *IEEE Pervasive Computing*, volume 8(4), October 2009.
17. K. Nahrstedt X. Gu, A. Messer, I. Greenberg, and D. Milojicic. Adaptive offloading inference for delivering applications in pervasive computing environments. In *Proceedings of IEEE International Conference on Pervasive Computing and Communications(PerCom)*, Dallas-Fort Worth, Texas, March 2003.