2011 10th International Symposium on Parallel and Distributed Computing

Augmented Mobile Devices through Cyber Foraging

Somayeh Kafaie, Omid Kashefi, and Mohsen Sharifi
School of Computer Engineering
Iran University of Science and Technology
Tehran, Iran
so_kafaie@comp.iust.ac.ir, kashefi@{ieee.org, iust.ac.ir}, msharifi@iust.ac.ir

***Abstract*— There is increased demand to run applications on mobile devices in recent years. Although many of these alluring applications are resource-intensive, they expect to get the same performance on mobile devices as on powerful non-mobile computers. On the other hand, considerations such as weight, size, and mobility impose constraints on mobile devices and restrict their processor speed, memory size and battery lifetime. Cyber foraging ameliorates this performance disparity by utilizing nearby non-mobile computers called surrogates to run the whole or parts of applications, which are offloaded from mobile devices. However, cyber foraging is not suitable for all circumstances. In this paper, we propose a mechanism to determine the best location, either a mobile device or best surrogate(s) around, to run an application by calculating the cost of offloading the task of running the application to each location according to the context's metrics such as specifications of mobile device, surrogates, network, and application. Experimental results show that our proposed mechanism almost always selects the best one between local execution of the application on the mobile device and task offloading to surrogate(s) according to the current input size of the application, with respect to latency and energy consumption.**

Keywords- Pervasive Computing; Cyber Foraging; Resource; Mobile Devices; Surrogate; Energy; Latency

I. INTRODUCTION

The number of computers per user is evolving from one mainframe for many users, through one personal computer for everyone, to many computing devices as PDAs, cell phones, and laptops for every single user in pervasive computing environments [1]. Pervasive computing environment is introduced by Mark Weiser [2] as an environment which is augmented with computing resources and can provide information and services for users anytime and anywhere. Due to the important role of users in such environments, one of the most important keys of pervasive computing environments is mobility and using mobile devices [3].

On the other hand, users expect to run applications on mobile devices and get the same performance as running them on powerful static computers. However, many of these tempting applications are resource-intensive and hard to perform on mobile devices; speech recognizer [5, 6], language translator [5, 6], scientific calculator [4], and image manipulator [7, 8] are examples of such applications that may not perform on mobile devices at all or as good as on static computers.

These applications require high computing power, memory, and battery lifetime which make their execution on mobile devices impossible or along with low and undesirable performance. Considerations such as weight, size, and heat dissipation that forms the nature of mobile devices impose severe restrictions on computational resources such as processor speed, memory size and disk capacity [8] and mobile devices are always resource poor in contrast with static computers.

Satyanarayanan has introduced cyber foraging [9] or task offloading as a solution to resolve the challenges of executing resource intensive applications on resource constrained mobile devices in pervasive computing environments. In this approach, the idle static computers in the vicinity of mobile devices, called Surrogates, are used to execute the whole or some parts of resource-intensive application on behalf of the mobile device. However, there is a challenge: "Is cyber foraging approach reasonable in all situations?"

To benefit from cyber foraging, some application related data, such as input parameters and application codes, must be sent from the mobile device to the surrogate and also the output results should be received. It is evident that if the cost of data transmission (i.e. communication cost) is more than the cost of local execution or computation cost, task offloading is not beneficial and local execution of application is superior to remote execution by cyber foraging. Therefore, an effective task scheduling and selecting the best location to run the task is needed to manage the trade-off between communication and computation cost of offloading the task.

In this paper, we exploit the cyber foraging approach to improve the performance and battery lifetime on mobile devices. We introduce a mechanism to determine the best location to run the task that would be mobile device (i.e. not to offload the task) or best surrogate(s) around by calculate the cost of offloading the task to each location according to the context's metrics such as mobile device, surrogates, network, and application specifications.

The remainder of the paper is organized as follows. Section II presents our proposed mechanism and describes its implementation detail. The results of experimental evaluations are depicted in Section III. Related researches on task offloading are discussed in Section IV, and Section V concludes the paper.

II. Proposed Mechanism

To augment the resources of mobile devices, we introduce a solver that is responsible for task scheduling and select the best location to run a task by calculating the cost of offloading for each location. However, there are three problems that should be resolved: 1) what is defined as the offloading cost? 2) what are the most important parameters affect the cost function? 3) how to calculate the offloading cost before real execution of the task?

When a task is going to run, the solver calculates the cost for every available machine (i.e. the mobile device and surrogates) that satisfies two following conditions:

a) Machine's available memory is more than required memory of the task.
b) The energy consumption of the mobile device to execute the task on this machine is less than available energy of mobile device.

Then the solver makes a decision to offload the task to one of the nearby surrogates or runs the task locally. Actually, the solver chooses a location that has the minimum cost to run the task and also satisfies two mentioned conditions. Figure 1 shows an overview of our proposed mechanism.

A. What is Defined as the Offloading Cost

One of the most important constraints of mobile devices is energy consumption because energy cannot be replenished [10] and considerations such as weight and size of mobile devices limit battery lifetime. Nevertheless, nowadays users run more energy-intensive applications on the mobile devices [11].

Therefore, we have considered energy consumption as an effective factor to make decision about the offloading.

On the other hand, latency is an important factor for users and as the programs usually execute on the mobile device by user's demand, the preferences of the user are another important metric. We define latency as the delay time between receive of application's input from the user and presentation of application's output to the user.

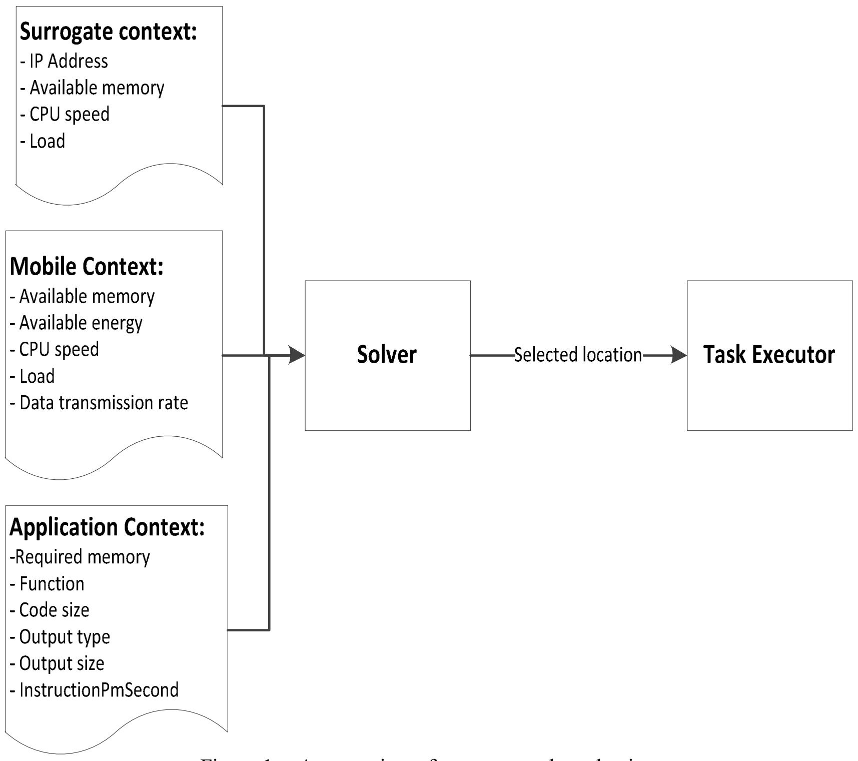

Figure 1. An overview of our proposed mechanism

Therefore, we define the offloading cost function to improve two important factors of mobile devices, energy consumption and latency as (1). We calculate the cost value for each machine including surrogates and mobile device itself and the machine with the minimum cost value would be the offloading target.

$$Cost = w_1 * Latency + w_2 * Energy \tag{1}$$

If cyber foraging causes a decrease in both latency and energy consumption, task offloading is effective and desirable. On the contrary, an increase in both latency and energy consumption may shows that task offloading is not beneficial. However, there are some cases that the variation of these two factors is on the opposite way. Selecting the best location in these cases is not straightforward and needs some considerations. For example, when battery life-time is low, solver should choose a location which minimizes energy consumption. On the other hand, for applications wherein time and latency is very important, latency cost is superior to energy cost.

In our implementation, we assume if energy level of the mobile device is less than a defined threshold, solver selects the best location to minimize the energy consumption which means $\mathrm{w}_1 = 0$ and $\mathrm{w}_2 = 1$. In other cases, the weight of latency and energy in offloading cost function is equal (i.e. $\mathrm{w}_1 = \mathrm{w}_2 = 0.5$).

B. Offloading Metrics

We categorize the most important context information which influences the offloading cost function into three classes, as follows:

- *Mobile and surrogate specifications* include processor speed, load and available memory of the mobile device and surrogates and available battery lifetime of the mobile device.
  To take into account current processing power (the combination of processor speed and load) of every machine to execute every task we define *InstructionPmSecond* parameter that will be described in Section II.C.1.
- *Application specifications* include application code, input and output size. Furthermore, some parts of applications are not transferable to the surrogate, e.g. codes that interact with I/O devices [11-13], and native methods of a language with different implementations on different platforms [1]. Because application code and input are available before calculating offloading cost, their size can be specified by the solver. Although output size is not available before task execution, in most cases it is a constant value with a definite size or it can be estimated in terms of input value or input size.
- *Network specifications* include type and bandwidth of communication network.
  Due to some reasons that will be discussed in Section II.C.2, network information used by the solver is constant and is measured in advance.

Solver uses these specifications in 3 context descriptor file as mobile context, surrogate context and application context according to Figure 1.

C. Latency Estimation

We define latency as the delay between receiving the application's input from the user and delivering the application's output to the user. Our proposed solver estimates the latency for two different cases:

a) *Remote execution* that means a surrogate is candidate for execution location.
b) *Local execution* or executing the task on the mobile device.

We can calculate latency for both mentioned cases by (2).

$$Latency = Time_{computation} + Time_{communication} \tag{2}$$

1) Computation Time

Computation time, indicated by $\mathrm{Time}_{\mathrm{computation}}$ in (2) is equal to execution time of the application on each location (i.e. the mobile device and surrogates).

Almost all previous works use online profiling to estimate computation time. In fact, they monitor execution time of every task and generate either a simple linear model of application behavior [6, 11, 14, 15] or the average execution time of previous runs [1, 4, 16, 17] to estimate execution time of future runs.

In contrast with these researches, we use offline profiling. To estimate computation time, we define a *Function* for every task which is calculated simply according to the time complexity order of the task. For example, the value of *Function* for *finding matrix determinant* application is *N!*, where *N* is the row count or is N^2 for *selection sort*, where *N* is the array size. Furthermore we define *InstructionPmSecond* parameter for every task and machine that is calculated by (3).

$$InstructionPmSecond = \frac{Function(N)}{Execution\ Time} \tag{3}$$

Function parameter for every task has been specified by task developer in advance. On the other hand, *InstructionPmSecond* parameter is measured by offline profiling for every task and the mobile device and is available before task execution in application context descriptor. To calculate *InstructionPmSecond* parameter for surrogates, we use following method.

When the mobile device is going to run a task whose corresponding *InstructionPmSecond* parameter for some surrogates is not available (i.e. the surrogate was not profiled for that task before), we send a task profiler to surrogates. This profiler which has been generated by task developer in advance, contains application code with a comparatively small input data, runs the application on the surrogate, calculates *InstructionPmSecond* for the surrogate by (3), and sends back the calculated *InstructionPmSecond* parameter to the solver. Then this parameter will be saved in the context descriptor of corresponding surrogate for next uses.

Although calculating *InstructionPmSecond* in this way imposes a little latency to the system for the first run, the experimental results in Section III shows that the precious effect of this factor to have a good estimation of execution time can well cover the imposed overhead.

To calculate the execution time for corresponding input value on every machine, the solver calculates the *Function(N)* score for the input value of the application presented in the application context description by substituting input value for *N* parameter. The result is then divided by *InstructionPmSecond* of each machine to estimate the *execution time* of the application on each machine for the specific input value.

2) Communication Time

It is evident that communication time, indicated by $\mathrm{Time}_{\mathrm{communication}}$ in (2) is equal to zero, for *local execution* on the mobile device. But in *Remote execution* case, $\mathrm{Time}_{\mathrm{communication}}$ is calculated by (4) and includes two parts: 1) the time is taken to send related data such as input data and application code to the selected location, and 2) the time of receiving the output data from the selected location.

$$Time_{communication} = Time_{send} + Time_{receive} \qquad (4)$$

$\mathrm{Time}_{\mathrm{send}}$ and $\mathrm{Time}_{\mathrm{receive}}$ are calculated in terms of *Data Transmission Rate* and *Transmission Data Size* by (5).

$$Time_{Send/receive} = \frac{Transmission\ Data\ Size}{Data\ Transmission\ Rate} \qquad (5)$$

Transmission Data usually includes input, output and code of the task and we discussed their size earlier in Section II.B. On the other hand, there are some ways to estimate *Data Transmission Rate*. Transferring a sample file with specific size, online, is one possible solution [11]. Figure 2 shows transmission time of a 50 KB file from the mobile device to a surrogate in WLAN network, for 50 iterations.

As Figure 2 shows, results are various and *Data Transmission Rate* cannot be calculated using only one transfer time of a file. To have a reliable *Data Transmission Rate*, data should be repeatedly transmitted which generates large amounts of traffic on the network that is not desirable.

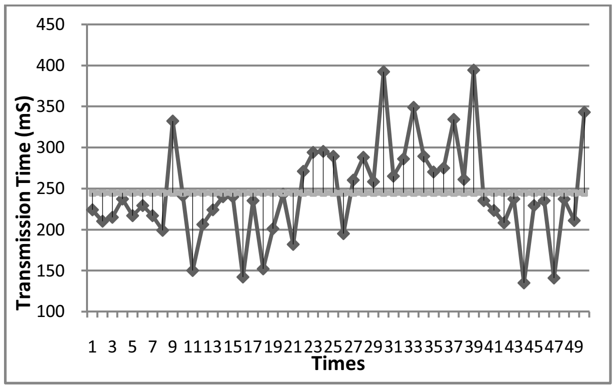

Figure 2. Transmission time of a 50 KB file between a mobile device and a surrogate

Using adaptive history-based approach is another possible solution. It means measuring *Data Transmission Rate* whenever data is transmitting between two nodes and using it for future estimations [6, 14]. Figure 3 shows the average measured *Data Transmission Rate* in 50 iterations for different file sizes.

According to Figure 3, the data is transferred faster for larger files. It is because of the nature of TCP connections. When the file size is growing, the sliding window is expanded to speed up the transfer [16]. Therefore, if for example, the measured *Data Transmission Rate* by transfer a 5 KB file is used to estimate transmission time of a 5 MB file, the calculated cost by the solver is not precise and it may cause a wrong decision about execution location.

According to the previous discussion, we used a third solution. We defined *Data Transmission Rate* as a constant value for different ranges of transmission data size. We have measured these values by offline profiling for an IEEE 802.11 b/g WLAN network that is used in our experiments. The results are described in Table I. Our experimental results in Section III show the constant value works good enough and also it does not impose any overhead on the system.

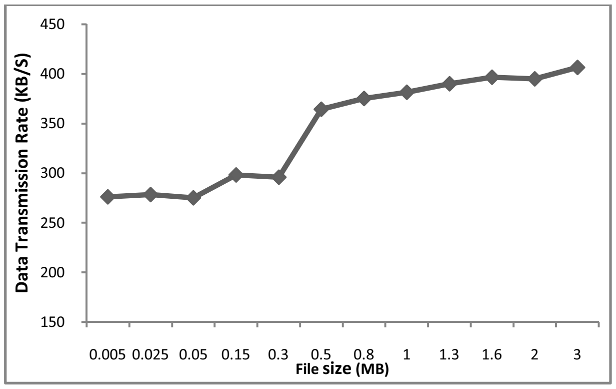

Figure 3. Data transmission rate for different file sizes

TABLE I. DATA TRANSMISSION RATE FOR DIFFERENT SIZES

Transmission Data Size (KB)	<350	>350 & <700	>700
Transmission Data Rate (KB/S)	280	350	390

D. Estimation of Energy Consumption

We estimated energy consumption of mobile device in various states (i.e. idle, computing, sending, and receiving) in terms of the time. We monitored energy consumption of mobile device in these states by PowerGuard V1.3 [18]. This software monitors the current and voltage of the mobile device and calculates a value as mAh, too. We estimated energy consumption rate by (6) to (9). Table II shows the result.

$$milli\ Power\ per\ Hour\ (mWh) = mAh * Voltage \quad (6)$$

$$Power\ per\ Hour\ (Wh) = mWh/1000 \quad (7)$$

$$Energy\ (joule) = Wh * 3600 = WS \quad (8)$$

$$Energy\ consumption\ rate = \frac{Energy(Joule)}{Time(Second)} \quad (9)$$

TABLE II. ENERGY CONSUMPTION RATE FOR THE MOBILE DEVICE

Wireless Network	**Mobile Status**	**Energy Consumption Rate (Joule/S)**
On	Idle	0.137
	Sending	0.975
	Receiving	0.78
	Computing	0.628
Off	Idle	0.123
	Computing	0.560

According to these energy rates, we define energy consumption of mobile device in local and remote execution cases by (10) and (11).

$$Energy_{mobile} = Time_{mobile} \times EnergyRate_{Computing} \quad (10)$$

$$\begin{aligned} Energy_{surrogate} = {} & (Time_{send} * EnergyRate_{Send}) \\ & + \left(ExecutionTime_{surrogate} * EnergyRate_{idle}\right) \\ & + (Time_{receive} * EnergyRate_{receive}) \end{aligned} \quad (11)$$

III. EXPERIMENTS

To quantify the effectiveness of our proposed mechanism, we construct a test bed and design several experimental scenarios. The test bed consists of one surrogate and one HTC Touch 2 smartphone. The mobile device is connected to the surrogate via IEEE 802.11b/g WLAN. Table III describes the detailed configurations of these machines.

We evaluate our proposed mechanism with regard to latency and energy consumption in 3 scenarios: 1) local execution where the mobile device executes the task itself, 2) remote execution or offloading where the mobile device sends application code and input data to a surrogate to execute the task on behalf of it, 3) using our proposed mechanism to find the best execution location according to the current situations and run it.

We evaluate the benefits of our proposed mechanism on 20 iterations of running two applications in terms of different inputs where user intends to run an application of *selection sort* of an array in terms of various array sizes and an application of *matrix determinant* in terms of different row counts. Both of these applications are CPU-intensive and the output size of the latter is a constant value, while the former has an output size equal to input size. Figure 4 and Figure 5 show used context descriptor files for these two applications.

```
<ApplicationContext>
        <Name> MatrixDeterminant </Name>
        <ID> 102 </ID>
        <RequiredMemory>
                5242880+(((4*N)^2)*2048)
        </RequiredMemory>
        <Function> N! </Function>
        <InstPms> 90 </InstPms>
        <CodeSize> 3072B </CodeSize>
        <OutputType> const </OutputType>
        <OutputSize> 500B </OutputSize>
</ApplicationContext>
```

Figure 4. Context specification of the *matrix determinant* application

TABLE III. CONFIGURATION OF DEVICES USED IN THE EXPERIMENTS

Type	**Processor**	**Memory**	**Operating System**
Mobile device	Qualcomm MSM7225™ 528 MHz	256 MB	Windows Mobile 6.5 Professional
Surrogate	Intel Core 2Duo 2.5 GHz	4 GB	Windows 7 Professional

```
<ApplicationContext>
            <Name> SelectionSort </Name>
            <ID> 105 </ID>
            <RequiredMemory>
                        5242880+(260*N)
            </RequiredMemory>
            <Function> N^2  </Function>
            <InstPms> 4400 </InstPms>
            <CodeSize> 2048B </CodeSize>
            <OutputType> size </OutputType>
            <OutputSize> N </OutputSize>
</ApplicationContext>
```

Figure 5. Context specification of the *selection sort* application

A. Latency

Latency is defined as the amount of time is taken to respond upon a user-triggered request. Usually, less latency causes more users' satisfaction. To evaluate the impact of our proposed mechanism on latency, we measured it on three mentioned scenario in 20 iterations. Figure 6 and 7 show the average latency for two applications of *matrix determinant* and *selection sort*, respectively. To emphasize on latency in this case, we set $w_1 = 1, w_2 = 0$ in (1).

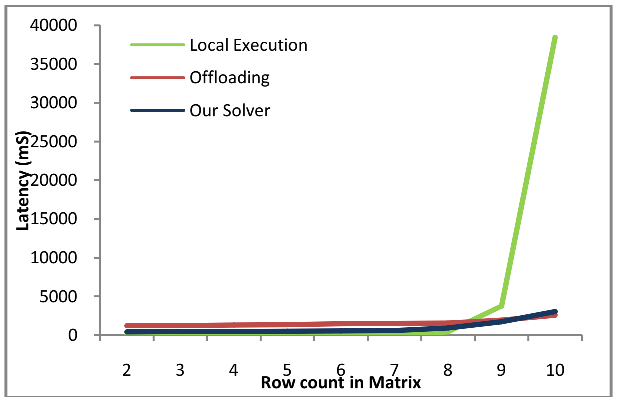

Figure 6. Latency comparison for *matrix determinant* application

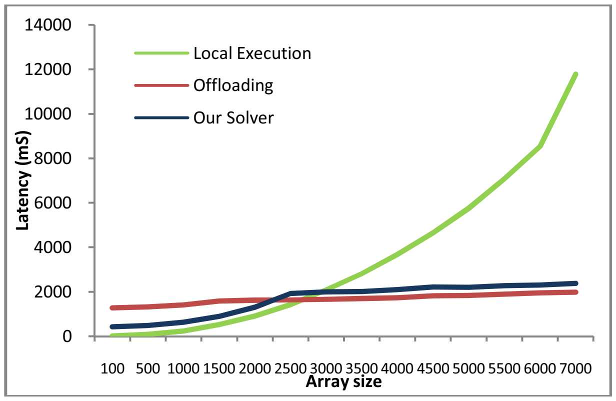

Figure 7. Latency comparison for *selection sort* application

B. Energy Consumption

In today's world, user demand to run resource intensive applications on mobile devices is increasing and one of the most important resources of mobile devices is energy. Cuervo *et al.* [11] show that how fast the battery of mobile devices is drained by running a resource intensive application that its usage on mobile devices is not unrealistic. Therefore, a good offloading mechanism should focus on consuming as low energy as possible.

We evaluate energy consumption of our proposed method by running the applications of *matrix determinant* and *selection sort* in three mentioned scenario (i.e. local execution, blind offloading, and using our solver). Figure 8 and Figure 9 show the result for this experiment. We set $w_1 = 0$, and $w_2 = 1$ in (1), due to emphasize on energy consumption. Also, in the case of local execution, the WiFi interface of the mobile device is turned off to preserve energy resources.

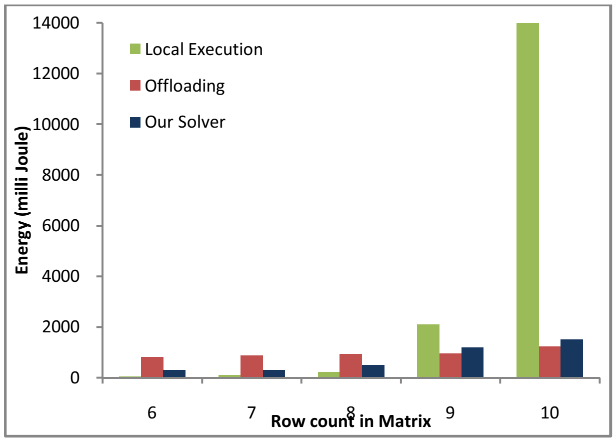

Figure 8. Comparison of energy consumption for *matrix determinant* application

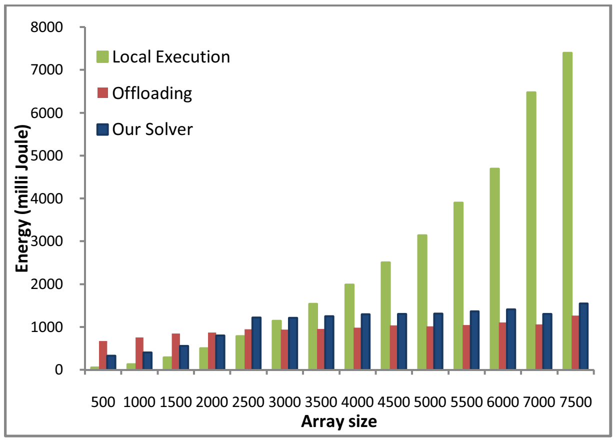

Figure 9. Comparison of energy consumption for *selection sort* application

IV. Related Work

The idea of cyber foraging in pervasive computing environments was introduced by Satyanarayanan [9]. Spectra [6, 15] and Chroma [5, 14] are the first cyber foraging systems that improve latency and energy consumption of mobile devices. They monitor available resources on the mobile device and nearby surrogates and use the results and monitored costs of previous executions to estimate the execution cost of future runs.

Gu *et al.* [1, 19] and Ou *et al.* [4, 13] have used a graph model to partition the application and select an execution location for every part. While Gu *et al.* partition applications according to the required memory of each part and available memory on every location, Ou *et al.* take into account CPU and bandwidth constraints, too.

Curvo *et al.* have introduced a cyber-foraging system called MAUI [11] and select the best location to execute a task according to the energy consumption. MAUI profiles energy consumption of the mobile device in previous executions of every task on every machine and presents a model of energy consumption to estimate it in next runs.

On the other hand, Kristensen has introduced Scavenger [16, 17], a cyber-foraging system to improve latency in mobile devices. Scavenger calculates the average of previous execution time of every task on every machine and saves them in some buckets according to the input value.

All mentioned works have used adaptive history-based approach and online profiling to estimate execution time on mobile device and surrogates. In fact, they use previous runs to estimate execution time on future ones. There are some shortcomings with this approach:

a) In first executions, there is no valid estimation of execution time to use by solver and solver's decisions are probably wrong.
b) The effect of input size on execution time is not considered or doesn't have enough precision.
c) History-based approach imposes dynamic profiling overhead to the system.

On the contrary, we don't need to save information of previous runs and we estimate execution time in terms of input size with high precision simply, by defining two factors as *Function* and *InstructionPmSecond* for every task and much less dynamic profiling overhead.

V. Summary

With ever increasing usage of mobile devices in today's life, users expect to run the same applications on mobile devices and static computers. However mobile devices have always suffered from resource constraints, in comparison with static computers, to run complex and high computational applications. Cyber foraging or task offloading is one of the most common solutions to resource poverty of mobile devices, especially in pervasive computing environments.

However, cyber foraging idea is not beneficial in all situations and there is a need to manage the trade-off between computation and communication cost of task offloading according to the current situations. To manage this trade-off, we calculated the offloading cost for the mobile device and every available surrogate. In order to select the best location to run a task, our proposed mechanism takes into account context metrics such as mobile device, surrogates, communication network, and application specifications.

Also, we introduced a good solution to have an acceptable estimation of execution time and energy consumption, before real execution. The experimental results show that our proposed solver almost always selects the best location to run a task that would be the mobile device, itself or a surrogate. our proposed mechanism, preserve nearly the same response time and energy consumption compared to blind offloading approach, when it decides to offload the task; and nearly the same response time and energy consumption compared to local execution on mobile device, when it decides to execute the task on mobile phone.

References

[1] X. Gu, A. Messer, I. Greenbergx, D. Milojicic, and K. Nahrstedt, "Adaptive Offloading for Pervasive Computing," *IEEE Pervasive Computing Magazine,* vol. 3, no. 3, pp. 66-73, July 2004.

[2] M. Weiser, "The Computer for the 21st Century," *Scientific American Special Issue on Communications, Computers, and Networks,* pp. 94-104, September 1991.

[3] L. Kolos-Mazuryk, R. Wieringa, and P. Van Eck, "Development of a Requirements Engineering Method for Pervasive Services," in *RE '05 Doctoral Consortium*, Paris,France, 2005.

[4] S. Ou, K. Yang, and Q. Zhang, "An Efficient Runtime Offloading Approach for Pervasive Services," in *IEEE Wireless Communications & Networking Conference (WCNC2006)*, Las Vegas, 2006, pp. 2229-2234.

[5] R. K. Balan, G. Gergle, M. Satyanarayanan, and J. Herbsleb, "Simplifying Cyber Foraging for Mobile Devices," in *5th USENIX International Conference on Mobile Systems, Applications and Services (MobiSys)*, San Juan, Puerto Rico, 2007, pp. 272-285.

[6] J. Flinn, S. Park, and M. Satyanarayanan, "Balancing Performance, Energy, and Quality in Pervasive Computing," in *22nd International Conference on Distributed Computing Systems (ICDCS'02)*, Vienna, Austria, 2002, pp. 217-226.

[7] G. Chen, B. T. Kang, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, and R. Chandramouli, "Studying Energy Trade Offs in Offloading Computation/Compilation in Java-Enabled Mobile Devices," *IEEE Transactions on Parallel and Distributed Systems,* vol. 15, no. 19, pp. 795-809, September 2004.

[8] M. Satyanarayanan, P. Bahl, R. Cáceres, N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," *IEEE Pervasive Computing,* vol. 8, no. 4, pp. 14-23, October-December 2009.

[9] M. Satyanarayanan, "Pervasive Computing: Vision and Challenges," *IEEE Personal Communication,* vol. 8, no. 4, pp. 10-17, August 2001.

[10] M. Satyanarayanan, "Avoiding Dead Batteries," *IEEE Pervasive Computing,* vol. 4, no. 1, pp. 2-3, January-March 2005.

[11] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: Making Smartphones Last Longer with Code Offload," in *8th international conference on Mobile systems, applications, and services (ACM MobiSys'10)*, San Francisco, USA, 2010, pp. 49-62.

[12] M. Othrnan, and S. Hailes, "Power Conservation Strategy for Mobile Computers Using load sharing," *Mobile Computing and Communications Review,* vol. 2, no. 1, pp. 19-26, January 1998.

[13] S. Ou, K. Yang, and A. Liotta, "An Adaptive Multi-Constraint Partitioning Algorithm for Offloading in Pervasive Systems," in *4th Annual IEEE International Conference on Pervasive Computing and Communications (PERCOM'06)*, Pisa, Italy, 2006, pp. 116-125.

[14] R. K. Balan, M. Satyanarayanan, S. Park, and T. Okoshi, "Tactics-Based Remote Execution for Mobile Computing," in *1st International Conference on Mobile Systems, Applications and Services*, San Francisco, 2003, pp. 273-286.

[15] J. Flinn, D. Narayanan, and M. Satyanarayanan, "Self-Tuned Remote Execution for Pervasive Computing," in *8th IEEE Workshop Hot Topics in Operating Systems*, Schloss Elmau, Germany, 2001, pp. 61-66.

[16] M. D. Kristensen, "Empowering Mobile Devices through Cyber Foraging:The Development of Scavenger, an Open Mobile Cyber Foraging System," PhD Thesis, Department of Computer Science, Aarhus University, Denmark, 2010.

[17] M. D. Kristensen, and N. O. Bouvin, "Scheduling and Development Support in the Scavenger Cyber Foraging System," *Pervasive and Mobile Computing,* vol. 1, no. 6, pp. 677-692, December 2010.

[18] *PowerGuard.* Available: http://www.vandenmuyzenberg.nl/PowerGuard/

[19] X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, and D. Milojicic, "Adaptive Offloading Inference for Delivering Applications in Pervasive Computing Environments," in *1st IEEE International Conference on Pervasive Computing and Communications (PerCom'03)*, Fort Worth, Texas, USA, 2003, pp. 107-114.