Context-Aware Task Scheduling for Resource Constrained Mobile Devices

Somayeh Kafaie, Omid Kashefi, and Mohsen Sharifi

School of Computer Engineering,
Iran University of Science and Thechnology,Tehran, Iran
so_kafaie@comp.iust.ac.ir,kashefi@{ieee.org,iust.ac.ir},
msharifi@iust.ac.ir

Abstract. Nowadays, mobile devices are very popular and accessible. Therefore users prefer to substitute mobile devices for stationary computers to run their applications. On the other hand, mobile devices are always resource-poor in contrast with stationary computers and portability and limitation on their weight and size restrict mobile devises' processor speed, memory size and battery lifetime. One of the most common solutions, in pervasive computing environments, to resolve the challenges of computing on resource constrained mobile devices is cyber foraging, wherein nearby and more powerful stationary computers called surrogates are exploited to run the whole or parts of applications. However, cyber foraging is not beneficial for all circumstances. So, there should be a solver unit to choose the best location either the mobile device or a surrogate to run a task. In this paper, we propose a mechanism to select the best method between local execution on the mobile device and remote execution on nearby surrogates to run an application by calculating the execution cost according to the context's metrics such as mobile device, surrogates, network, and application specifications. Experimental results show the superiority of our proposed mechanism compared to local execution of the application on the mobile device and blind task offloading with respect to latency.

Keywords: Pervasive Computing, Cyber Foraging, Mobile Computing, Task Offloading, Latency.

1 Introduction

These days, the users of mobile devices are increasing from day to day. On a planet with around 6.8 billion people, the number of people with cell phone subscriptions worldwide has reached 4.6 billion at the end of 2009 and is expected to reach five billion by the end of 2010 [1]. People all over the world are increasingly using their cell phones for daily tasks such as Internet banking, emailing, and emergencies.

However, mobile devices are always resource-poor. At any level of cost and technology, considerations such as weight, size, battery lifetime, ergonomics, and heat dissipation impose severe restrictions on computational resources such as processor speed, memory size and disk capacity [2]. Although mobile device technology is evolving but mobile devices always remain more resource constrained than traditional stationary computers [2, 3].

V. Snasel, J. Platos, and E. El-Qawasmeh (Eds.): ICDIPC 2011, Part I, CCIS 188, pp. 252–261, 2011.

Due to these restrictions, execution of some applications on mobile devices is impossible or along with low and undesirable performance. For example, imagine a person who has a complete and smart map on his mobile device and is going to find the best path to go to a destination. Scientific calculator [4], speech recognizer [5, 6], language translator [5, 6] and image manipulator [2, 7] are some more examples of such applications that their execution may need more energy or memory than available in mobile device or impose an unacceptable latency to the system and users.

Satyanarayanan has introduced cyber foraging [8] 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 nearby stationary computers, called Surrogates, are used to execute the whole or some parts of resource-intensive application on behalf of the mobile device. As computers become cheaper and more plentiful, cyber foraging approaches become more reasonable to employ in pervasive computing. However, there is a challenge: "Is cyber foraging approach reasonable in all situations?"

In this paper, we exploit cyber foraging approach to improve the performance on mobile devices. We introduce a solver unit that is responsible to determine the best location to run the task that would be the mobile device (i.e. local execution of the task) or best surrogate(s) around (i.e. remote execution of the task) by calculating the cost of executing the task on 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. Related works on cyber foraging are discussed in Section 2. Section 3 presents our proposed solver and describes its implementation detail. The results of experimental evaluations are depicted in Section 4, and Section 5 concludes the paper.

2 Related Work

There are several approaches with different objectives that have used the offloading of applications on resource constrained mobile devices in pervasive computing environments. Spectra [6] is one of the first proposed cyber foraging systems is focused on reducing the latency and energy consumption. Chroma [5, 9] tries to improve spectra's idea and reduce the burden on the programmers. Both of these works suppose the application is installed on the surrogates and there is no need to send the application code. But it is evident that this assumption decreases the flexibility and their approach does not work on new surrogates and tasks.

While, Gu *et al.* [10] have used a graph model to select offloading parts of the program to improve memory constraint of the mobile device, Ou *et al.* [4, 11] have used a similar method to address the CPU and bandwidth constraint, too. Song *et al.* [12, 13] has proposed a middleware architecture, called MobiGo for seamless mobility to choose the best available service according to the bandwidth and latency, and Kristensen [14] has introduced Scavenger as a cyber-foraging framework whose focus is on CPU Power of mobile devices and decreases latency.

All mentioned works have used adaptive history-based approach and dynamic profiling to estimate execution time on mobile device and surrogates and select the best location to execute a task. In fact, they use previous runs to estimate execution

time on future ones. Therefore, in first executions, there is no valid estimation of execution time to use by the solver and solver's decisions are probably wrong. Furthermore, the effect of input size on execution time is not considered or doesn't have enough precision, and also history-based approach imposes dynamic profiling overhead to the system.

On the contrary, as it will be shown in next Section, 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.

3 Proposed Solver

3.1 Necessity of the Solver

Cyber foraging combines the mobility of small devices with high computing capability and extensive resources of nearby static servers by offloading the tasks from mobile devices to surrogates for remote execution [6]. Nevertheless, there is a challenge. Is task offloading 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.

Let us consider T_M as the time of running a program on a mobile device and T_S as the time of running the program on a surrogate. If D_T is the size of input data and code, D_R is output size, and B is the network bandwidth, the offloading time can be defined as $T_{offload} = (D_T + D_R)/B + T_S$.

It is obvious that the offloading is effective only when T_M is more than $T_{offload}$. As usually $T_M >> T_S$, 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, a solver unit is needed to manage the trade-off between communication cost and computation cost of offloading the task, schedule the task, and select the best location to run it.

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 task execution for each location.

When a task is going to run, the solver calculates the cost for every available machine (i.e. the mobile device and surrogates). 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. Figure 1 shows an overview of the proposed solver's flowchart.

3.2 Definition of Execution Cost

To define the execution cost, we have to determine the goal of task offloading. In general, Cyber foraging tries to augment some resources of mobile devices in terms of effective metrics to achieve more efficient application execution. The most important factors that have been considered in offloading approaches are as follows:

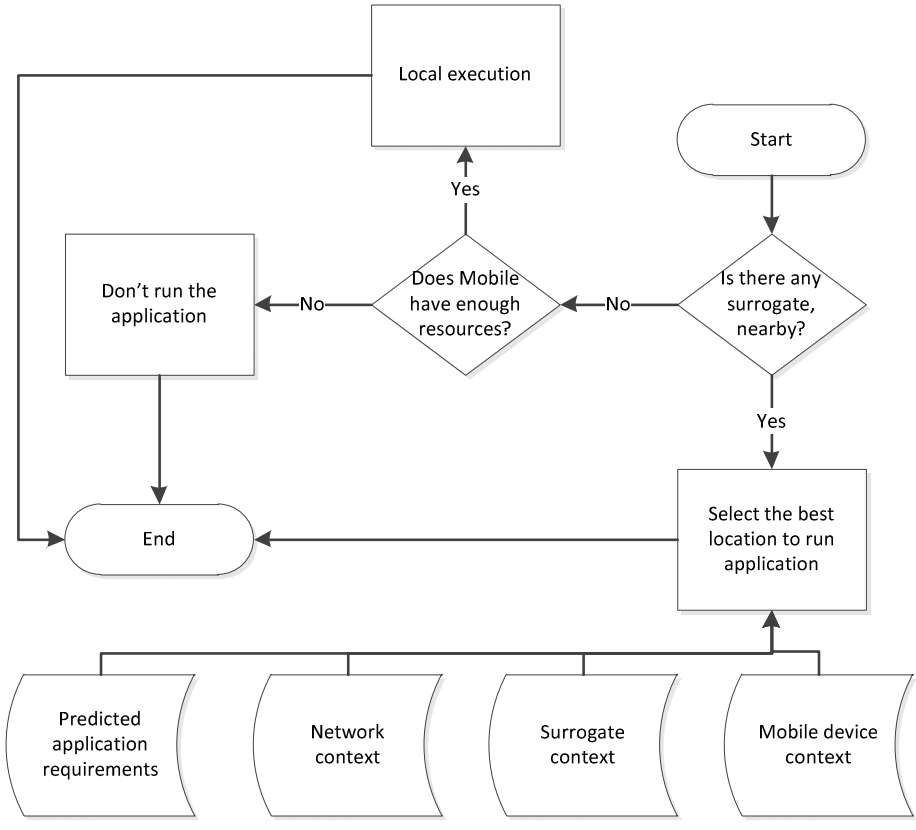

Fig. 1. An overview of our proposed solver's flowchart

- *Latency.* The processing power of mobile devices is considerably lower than static computers. Task offloading exploits powerful surrogates to decrease latency.
- *Energy*. One of the most important constraints of mobile devices is energy consumption because energy cannot be replenished [15]. Therefore the best location to run a task may be the one with minimum energy consumption.
- *Memory and storage*. Memory intensive applications cannot usually run on mobile devices and they need to be offloaded. Therefore execution cost can be calculated according to the required memory and storage of the tasks.

In this paper, we have defined execution cost according to the latency metric. Because latency is an important factor for users and as the programs usually execute on the mobile device by user's demand, the preference of the user is an 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.

Therefore, we define the execution cost function to improve an important factor of mobile devices, latency. 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.

3.3 Effective Parameters in Execution Cost

In this paper, the most important effective parameters which influence the execution cost function can be categorized into three classes, as follows:

- *Mobile and surrogate context* include current processor power of surrogates and mobile device.
- *Application context* 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, 16, 17], and native methods of a language with different implementations on different platforms [10].
- *Network context* include type and bandwidth of the communication network.

Solver uses this context information to calculate execution cost (latency) for every location and select the best one.

3.4 Latency Estimation

Latency is the delay between receiving the application's input from the user and delivering the application's output to the user. We estimate the latency or execution cost for two cases: (1) a surrogate is selected as execution location, and (2) local execution which means the mobile device runs the task itself.

The latency time in the mobile device is equal to execution time of the application on the mobile device but the latency of offloading the task to the surrogates includes three parts: (1) the time is taken to send related data such as input data and application code to the surrogate, (2) execution time on the surrogate, and (3) the time of receiving the output data from the surrogate. Equation 1 shows the latency of offloading the task to the surrogates.

$$\text{Latency}_{\text{surrogate}} = \text{Time}_{\text{send}} + \text{ExecutionTime}_{\text{surrogate}} + \text{Time}_{\text{receive}} \quad (1)$$

$\text{Time}_{\text{send}}$ and $\text{Time}_{\text{receive}}$ are calculated in terms of *Data Transmission Rate* and *Transmission Data Size* by Equation 2.

$$\text{Time}_{\text{Send/receive}} = \frac{\text{Transmission Data Size}}{\text{Data Transmission Rate}} \quad (2)$$

We defined *Data Transmission Rate* as a constant value and the results in Section 4 show the constant value works good enough. On the other hand, *Transmission Data* usually includes input, output and code of the task. The size of code and input is available, because making decision is just before task execution. Output size is usually a constant value with a definite size or it can be estimated in terms of input value or input size.

To estimate execution time, we defined a *Function* for every task which is calculated simply, according to the time complexity order of the task. For example, the value of *Function* is *N!* for *finding matrix determinant* application,where *N* is the row count, or is N^2 for *selection sort* application, where *N* is the array size. We calculated *InstructionPmSecond* value for every task and machine as Equation 3.

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

This information is measured for every task and the mobile device, in advance and is available before task execution in application context descriptor. To calculate the execution time for different input values on mobile device, solver calculates the Function score for the input value of the application presented in the application context descriptor. The result is then divided by *InstructionPmSecond* of mobile device which is also presented in application context descriptor to estimate the *execution time* of the application for the specific input value.

To estimate execution time on a surrogate, if the *InstructionPmSecond* was not presented for the corresponding application and surrogate (i.e. the surrogate was not profiled for that application before), we need to estimate *InstructionPmSecond* for that surrogate. Therefore, we generated a task profiler for the application which is sent to the surrogate to run the application. This profiler contains application code with a comparatively small input data, runs the application on the surrogate, calculates *InstructionPmSecond* for the surrogate by Equation 3, and sends back the calculated *InstructionPmSecond* factor to the solver. It takes about one second time for the tasks which are introduced in *Experiments* Section of this paper.

Although calculating *InstructionPmSecond* in this way imposes a little latency to the system for the first run, the experimental results in Section 4 shows that the precious effect of this factor to have a good estimation of execution time can well cover the imposed overhead. Also, by using this factor there is no need to monitor and save the execution time of task runs, as it was discussed in *Related Work* Section in more details.

4 Experiments

4.1 Experimental Platform

For our experiments, we used one HTC Touch 2 smartphone as the mobile device with a QualcommMSM7225™528 MHz processor and 256 MB RAM running Windows Mobile 6.5 Professional. The surrogate in this platform is a laptop running Windows 7 Professional. The surrogate's processor is Intel(R) Core(TM)2 Duo 2.5 GHz and its memory size is 4 GB. The mobile device is connected to the surrogate via 802.11b/g WLAN.

4.2 Test Applications

To quantify our proposed solver we used several applications that all of them are almost CPU-intensive tasks. We categorized these applications into three categories according to their usage, as follows: (1) scientific calculator operations such as calculating *matrix determinant*, (2) statistical applications such as sorting an array (*selection sort*), (3) daily applications such as *finding shortest path* between a source and a destination node in a map. Table 1 describes the detailed information about these applications.

The second and third columns of Table 2 (i.e. Function and InstructionPmSecond) have been used to calculate execution time on the mobile device by Equation 3. Output size is constant or is calculated in terms of input size and "N Description" column describes the *N* value in the *Function*.

Table 1. Detailed information about test applications

Name	Matrix determinant	Selection sort	Shortest Path
Function	N!	N^2	N^3
InstructionPmSecond	90	4400	550
Output Size	500B	INPUT	Sqrt (INPUT)
Output Type	Constant	Variable	Variable
N Description	Row count	Array size	Node count

4.3 Latency Evaluation

Latency is defined as the amount of time is taken to respond upon a user-triggered request. Usually, less latency causes more users' satisfaction. We evaluated our proposed mechanism with regard to the latency in 3 scenarios as follow.

a) Local execution where the mobile device executes the task itself
b) Remote execution or blind offloading where the mobile device always sends application code and input data to a surrogate to execute the task on behalf of it (without managing the trade-off between computation and communication costs)
c) Using our proposed mechanism to find the best execution plan between local execution (scenario 1) and remote execution (scenario 2) according to the current situations and input data of the task and run it.

We evaluated the benefits of our proposed mechanism on 20 iterations of running mentioned applications in terms of different inputs. We measured latency on three mentioned scenario in 20 iterations. Figure 2-4 show the average latency for three applications of *matrix determinant*, *selection sort*, and *shortest path* respectively.

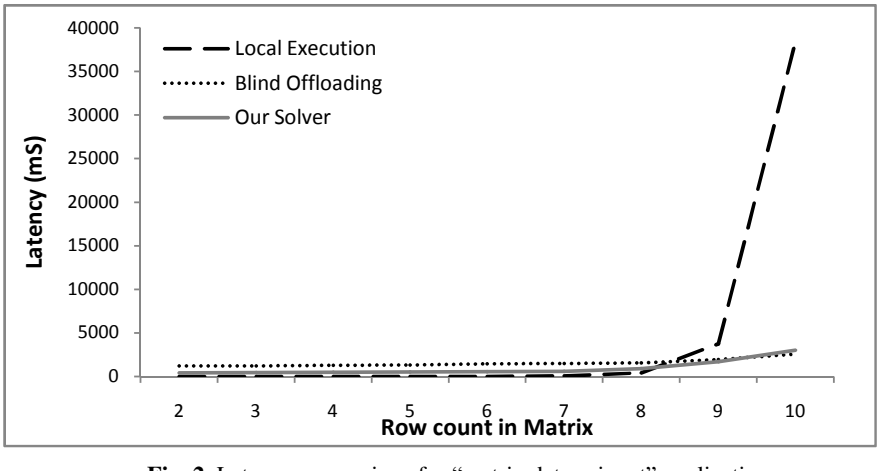

Fig. 2. Latency comparison for "matrix determinant" application

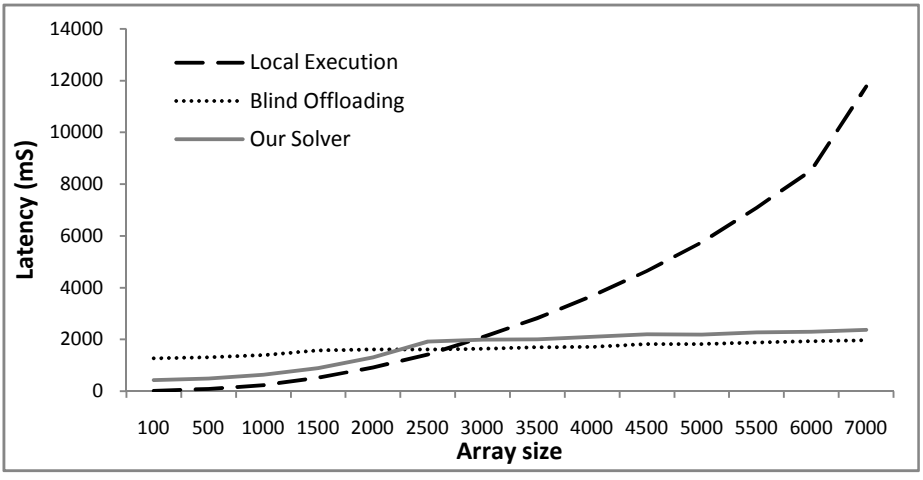

Fig. 3. Latency comparison for "selection sort" application

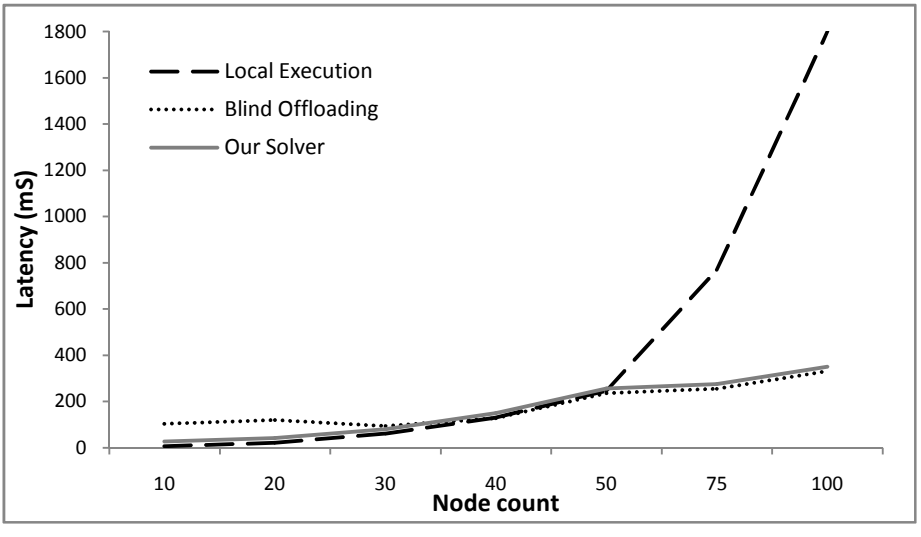

Fig. 4. Latency comparison for "shortest path" application

As it is clear in Figures 2-4, local execution on the mobile device is the best choice for small input values, due to communication overhead. On the contrary, for larger input values, remote execution (task offloading) is a better choice. Our proposed mechanism is almost always able to choose the best one between local execution and remote execution in terms of input value, with a small overhead.

5 Conclusion

These days, due to popularity of mobile devices, users intend to run more complex tasks on them and they expect to get the same performance as if they used to run their applications on powerful stationary computers. But it is evident that mobile devices are always resource poor in contrast with non-mobile computers.

Cyber foraging is one of the most common solutions for resource constraint in mobile devices. It augments mobile devices' capabilities by task offloading to nearby idle surrogates. Although powerful surrogates with more processing power and resources run tasks faster than resource constrained mobile devices, task offloading imposes communication overhead on the system. Because task information (e.g. code and input data) should be sent to the surrogate and output results should be received. Therefore task offloading is not beneficial in all situations.

In this paper, we meant to decrease latency by choosing the right one between local and remote execution. We proposed a solver unit that estimates latency of running the task on each location (i.e. the mobile device and surrogates) and selects the best location to run the task according to the context information such as processing power of every machine, code, input and output size of the task and network information.

Also, we introduced a good solution to have an acceptable estimation of execution time, before real execution. The experimental results show that our proposed mechanism is superior to local execution and blind task offloading in respect of latency.

References

1. The Global Partnership for Development at a Critical Juncture. United Nations, New York, MDG GAP Task Force Report (2010)
2. Satyanarayanan, M., Bahl, P., Cáceres, R., Davies, N.: The Case for VM-Based Cloudlets in Mobile Computing. IEEE Pervasive Computing 8, 14–23 (2009)
3. Oh, J., Lee, S., Lee, E.-s.: An Adaptive Mobile System Using Mobile Grid Computing in Wireless Network. In: Gavrilova, M.L., Gervasi, O., Kumar, V., Tan, C.J.K., Taniar, D., Laganá, A., Mun, Y., Choo, H. (eds.) ICCSA 2006. LNCS, vol. 3984, pp. 49–57. Springer, Heidelberg (2006)
4. Ou, S., Yang, K., Zhang, Q.: An Efficient Runtime Offloading Approach for Pervasive Services. In: IEEE Wireless Communications & Networking Conference (WCNC 2006), Las Vegas, pp. 2229–2234 (2006)
5. Balan, R.K., Gergle, D., Satyanarayanan, M., Herbsleb, J.: Simplifying Cyber Foraging for Mobile Devices. In: 5th USENIX International Conference on Mobile Systems, Applications and Services (MobiSys), San Juan, Puerto Rico, pp. 272–285 (2007)
6. Flinn, J., Park, S., Satyanarayanan, M.: Balancing Performance, Energy, and Quality in Pervasive Computing. In: 22nd International Conference on Distributed Computing Systems (ICDCS 2002), Austria, pp. 217–226 (2002)
7. Chen, G., Kang, B.T., Kandemir, M., Vijaykrishnan, N., Irwin, M.J., Chandramouli, R.: Studying Energy Trade Offs in Offloading Computation/Compilation in Java-Enabled Mobile Devices. IEEE Transactions on Parallel and Distributed Systems 15, 795–809 (2004)

8. Satyanarayanan, M.: Pervasive Computing: Vision and Challenges. IEEE Personal Communication 8, 10–17 (2001)
9. Balan, R.K., Satyanarayanan, M., Park, S., Okoshi, T.: Tactics-Based Remote Execution for Mobile Computing. In: 1st International Conference on Mobile Systems, Applications and Services, San Francisco, pp. 273–286 (2003)
10. Gu, X., Messer, A., Greenbergx, I., Milojicic, D., Nahrstedt, K.: Adaptive Offloading for Pervasive Computing. IEEE Pervasive Computing Magazine 3, 66–73 (2004)
11. Ou, S., Yang, K., Liotta, A.: An Adaptive Multi-Constraint Partitioning Algorithm for Offloading in Pervasive Systems. In: 4th Annual IEEE International Conference on Pervasive Computing and Communications (PERCOM 2006), Pisa, Italy, pp. 116–125 (2006)
12. Song, X.: Seamless Mobility in Ubiquitous Computing Environments. PhD Thesis, Georgia Institute of Technology (2008)
13. Song, X., Ramachandran, U.: MobiGo: A Middleware for Seamless Mobility. In: 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2007), Daegu, pp. 249–256 (2007)
14. Kristensen, M.D.: 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)
15. Satyanarayanan, M.: Avoiding Dead Batteries. IEEE Pervasive Computing 4, 2–3 (2005)
16. Othrnan, M., Hailes, S.: Power Conservation Strategy for Mobile Computers Using Load Sharing. Mobile Computing and Communications Review 2, 19–26 (1998)
17. Cuervo, E., Balasubramanian, A., Cho, D.K., Wolman, A., Saroiu, S., Chandra, R., Bahl, P.: MAUI: Making Smartphones Last Longer with Code Offload. In: ACM MobiSys, San Francisco, USA, pp. 49–62 (2010)