A Survey and Taxonomy of Cyber Foraging of Mobile Devices

Mohsen Sharifi, Somayeh Kafaie, and Omid Kashefi, *Member, IEEE*

***Abstract*—With the ever-increasing advancement of mobile device technology and their pervasive usage, users expect to run their applications on mobile devices and get the same performance as if they used to run their applications on powerful non-mobile computers. There is a challenge though in that mobile devices deliver lower performance than traditional less-constrained and non-mobile computers because they are constrained by weight, size, and mobility in spite of all their advancements in recent years. One of the most common solutions that has ameliorated this performance disparity is cyber foraging, wherein nearby non-mobile computers called surrogates are utilized to run the whole or parts of applications on behalf of mobile devices. In this paper, we present a survey of cyber foraging as a solution to resolve the challenges of computing on resource-constrained mobile devices. We also explain the most notable cyber foraging systems and present a categorization of existing cyber foraging approaches considering their type of dynamicity, granularity, metrics used, surrogate types and scale, location of their decision maker unit, remoteness of execution, migration support, and their overheads.**

***Index Terms*—Cyber Foraging, Mobile Devices, Resource-Constrained Computing, Taxonomy.**

I. Introduction

NOWADAYS, mobile devices are very popular. 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 such as viewing online traffic map or using routing applications to find the best next course or connecting to a medical information system to take a prescription urgently [2]. In this paper, by mobile device, we refer to pocket-sized handheld computing devices such as PDAs and Tablets with Wi-Fi connection, as well as to Smartphones that in addition to Wi-Fi connection are equipped with mobile broadband network technologies such as GPRS, EDGE, 3G, 4G, EV-DO, LTE and WiMAX.

With mobile computing and wireless Internet, the dream of accessing information anywhere and anytime is getting closer to reality [3]. However, mobile devices are always resource poor. At any level of cost and technology, considerations such as weight, size, battery life, ergonomics, and heat dissipation impose severe restrictions on computational resources such as processor speed, memory size and disk capacity [4]. Although mobile device technology is evolving but mobile devices always remain more resource constrained than traditional non-mobile computers [4], [5].

On the other hand, new applications running on mobile devices in recent years have attracted users to use and benefit from mobile devices. Examples of these applications include natural language translators [6], [7], speech recognizers [6]–[9], optical character recognizers [6], image processors [10]–[12], games with high computing, capture, edit, annotate and upload videos [13], and a useful application that helps Alzheimer people in their daily life by providing them with a wearable device with a head-up display in the form of eyeglasses, a camera for scene capture and earphones [4]. Unfortunately, these applications require higher computing power, memory, and battery lifetime than is available on resource constrained mobile devices. They also require faster responses than is currently supported on mobile devices.

Several approaches have proposed to empower the resource shortage of mobile devices. One approach is to rewrite applications anew for resource-constrained mobile devices. This approach is very expensive and can lead to ad-hoc applications [14], [15]. Two other approaches [15]–[20] have dealt with the problem of resource consumption, especially to increase battery lifetime.

The first approach addresses the issue from supply side (1) by manufacturing of more powerful resources (e.g. batteries with higher lifetime) while preserving their lightweight and small size [20] and (2) by replenishing a battery's energy by external actions such as human movement [15], [20] or by taking advantage of available energy resources such as solar power [16], [20]. Unfortunately, neither of these alternatives has remedied the resource consumption problem of resources noticeably. For example, in the battery lifetime prolonging case, the energy densities of batteries are already very high [15] and other alternatives though attractive have not been widely applicable and used yet.

The second approach tries to reduce the amount of required resources [15], [16]. The most favorable techniques in this approach include: (1) hardware and software management techniques, (2) fidelity adaptation [17], and (3) cyber foraging [19]. The hardware and software management techniques include techniques such as dynamic voltage frequency scaling [18] or similar ways to improve hardware power efficiency, or developing and deploying resource-aware software. Fidelity adaptation manages the trade-off between resource consumption and application quality where by fidelity we

Manuscript received 29 January 2011; revised 15 June 2011, 12 September 2011, and 06 October 2011.

The authors are with the School of Computer Engineering, Iran University of Science and Technology, Tehran, Iran (e-mails: msharifi@iust.ac.ir, so_kafaie@comp.iust.ac.ir, kashefi@{ieee.org, iust.ac.ir}). Mohsen Sharifi is the corresponding author.

Digital Object Identifier 10.1109/SURV.2011.111411.00016

1553-877X/12/$31.00 © 2012 IEEE

mean "an application-specific metric of quality that one can adjust by modifying the application's runtime parameters" [16]. Although the fidelity adaptation technique decreases the quality of results, it enables the execution of applications when there are no other solutions to run them in standard mode. Cyber foraging is beneficial when there are some idle stationary computers nearby mobile devices all connected via a wireless network such that the tasks of mobile devices can be offloaded to nearby surrogates to ameliorate resource poverty of mobile devices.

In this paper, we focus on the cyber foraging technique in the second approach. In what follows, Section II presents an overall view of different cyber foraging techniques. Section III presents the most effective metrics in cyber foraging techniques. Section IV presents and discusses the most popular available cyber foraging techniques. Section V presents our proposed taxonomy of cyber foraging techniques and Section VI concludes the paper.

II. Cyber Foraging

The term *"cyber foraging"* was first introduced by Satyanarayanan [19] to augment the computing resources of a wireless mobile device by exploiting available static computers, although similar offloading approaches to decrease the energy consumption of mobile devices had been proposed earlier by researchers such as Othrnan *et al.* [21]. Cyber foraging is the discovery of static idle computers called *surrogates* in the vicinity of a mobile device and entrusting some of the tasks of the mobile device to them [19]. As computers become cheaper and more plentiful, cyber foraging approaches become more reasonable to employ in pervasive computing [6], [7], [9], [14], [22]–[25], Grid computing [5], [26]–[28], and cluster computing [29].

In recent years, Cloud computing has also been used for some cyber foraging scenarios too [13], [30]–[38]. Although mobile devices can considerably benefit from offloading their tasks to computational Clouds, but there are some unresolved challenges in employing computational Clouds as surrogates for cyber foraging. There is no guarantee of availability of surrogates and application service level in computational Clouds [38]. Generally, users must pay for the Cloud services [23], [38]. To use Cloud services, mobile devices must be connected implying that applications become inaccessible when mobile devices are offline [35]. In addition, the use of 3G as the default connectivity solution for mobile devices, despite all improvements in broadband technologies, is still outperformed by WLAN in both energy consumption and network bandwidth and latency [36], [37], [39]. The lower bandwidth and the higher latency of 3G compared to WLAN are not solely due to the characteristics of these technologies. The disparity is attributed more to the transmission media they use. In 3G or other mobile broadband technologies, data is transmitted through the Internet in non-dedicated channels with higher communication latencies than in more dedicated channels of LANs that are mostly used by WLANs. Nevertheless, there is no substantial difference between Wi-Fi and 3G when data must be transmitted through the Internet [36], [37].

Anyhow, in some scenarios such as when there is no surrogate in the vicinity of mobile devices, the employment of cyber foraging in computational Clouds would be useful [4], [9], [40].

A. Cyber Foraging Qualification

Cyber foraging combines the mobility of small devices with high computing capability and extensive resources of nearby static servers by offloading the tasks of mobile devices to surrogates for remote execution [7]. Nevertheless, there is a challenge. Is offloading reasonable in all situations?

If enough resources (memory, energy, or storage) are not available on mobile devices to run a program, a decision must be made to offload or not to offload the program to nearby surrogates, based on the availability of resources on surrogates and the amount of resources required for offloading.

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. Let us further assume that the offloading requires transmission of D_T bytes of data and code, and receipt of D_R bytes of result; B_T is the network's transmission bandwidth and B_R is the network's receive bandwidth. We can define $T_{offload} = D_T/B_T + D_R/B_R + T_S$ as the offloading time. It is obvious that the offloading is effective only when T_M is bigger than $T_{offload}$. Therefore, as usually $T_M \gg T_S$, the computation part of program must be significantly larger than its communication part. This implies that when one uses cyber foraging to improve response time or energy consumption, the offloading mechanism would be more effective for applications requiring more computation than communication such as chess game and generation of very large prime numbers.

The goal of cyber foraging is to decrease the total response time (cost) of a program, not a part of it. For example, suppose running a compute intensive program on a mobile device with a large amount of data such as looking for the most similar picture to a given picture. Computation on the surrogate takes lower time, but on the other hand, the offloading of data takes a lot of time depending on the network type and distance between the surrogate and the mobile device. We illustrate a simple flowchart of cyber foraging qualification in Figure 1.

Therefore, we can conclude that large tasks requiring higher execution times make offloading more effective because the benefits of computation on a more powerful and faster surrogate outweigh the cost of communication. However, the constraints on mobile devices are due to the portability and mobility of such devices. The portability of mobile devices restricts the size of tasks [23], [41]. If a task is too large to complete its execution before leaving the area in the networking coverage of a surrogate, offloading is almost useless or complex and time-consuming solutions such as check pointing and process migration must be used, too. Therefore, a suitable offloading approach must specially consider the mobility nature of mobile devices and manage a trade-off between mobility and task size.

B. Cyber Foraging Steps

We summarize the steps of a cyber foraging approach as follows. It must be noted that all of the researches and works in

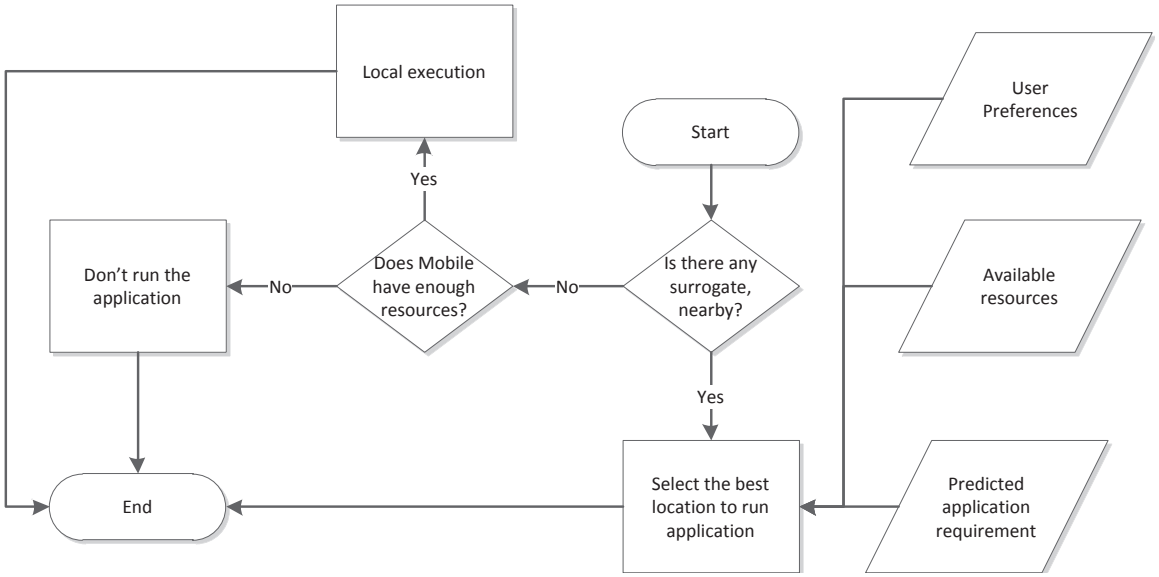

Fig. 1. A sample flow of running an application on a mobile device

cyber foraging and task offloading may not contain or address all of the following steps.

- **Surrogate Discovery.** To do cyber foraging, available idle surrogates must be found first. Some researches [14], [42]–[44] have addressed surrogate discovery.
- **Context Gathering.** Monitoring of available resources in surrogates and mobile devices and estimating application resource consumptions, defied as context gathering, are considered in some cyber foraging systems [6], [7], [23].
- **Partitioning.** In this step, a task is divided into smaller size subtasks, and undividable i.e. unmovable parts are specified. Some researches [30], [45] do the partitioning automatically.
- **Scheduling.** The most important step is making decision to place each task at the surrogate(s) most capable of performing it, based on the context information and the cost of doing so. Many researches [6], [7], [14], [21], [26], [38], [46]–[48] have considered this step.
- **Remote Execution Control.** The final step involves the establishment of a reliable connection to surrogate to pass its required information, remote execution, and the receipt of returned results. Various researches [4], [7], [9], [11], [23], [42], [49] have considered remote execution control.

III. Metrics of Cyber Foraging

In this section, we review the most effective metrics that can be used to decide whether to offload a program from a mobile device to one or more surrogates or not. We have categorized the cyber foraging metrics (Figure 2) into four groups namely: mobile and surrogate specifications, application specifications, network specifications, and context specifications that are discussed in follow.

A. Mobile and Surrogate Specifications

Different computers have different processor types, speeds, memory capacities, or storage sizes. If a mobile device does not have enough memory or storage to run a program, or its processor speed is too low and running the program takes too long, cyber foraging becomes a reasonable choice. In computational Clouds, the cost of surrogating is an important metric [50]. The cost of surrogating is measured in terms of processor cycles, memory size, storage size, communication traffic rate, input data size, and execution time of the chosen surrogate [35].

One of the most important reasons to offload a task is to reduce the energy consumption. The amount of usage of processor cycles, memory and storage for computation and communication with outer world via I/O devices, are the metrics that affect the energy consumption.

B. Application Specifications

Applications can be processor intensive, memory intensive, or I/O intensive [51]. As noted in Section II, cyber foraging is more useful to processor intensive applications; irrespective of mobility, higher rates of execution time scale for more offloading. Therefore, an effective metric for cyber foraging of applications is the intensity of computations and long execution times.

There may be some exceptions to offloading. It is probable that some parts of applications are not transferable to surrogates. These include codes that run local services such as user interfaces, codes that interact with I/O devices [21], [31], [47], codes that interact with external components that might be affected by re-execution [31], native methods of a language with different implementations on different platforms [14], parts that directly access device-specific information [47], tasks that need local resources to run [21], and components whose execution locations depend on other parts [52]. Therefore, the nature of an application is another important metric that determines which (parts of) applications can be offloaded and remotely executed.

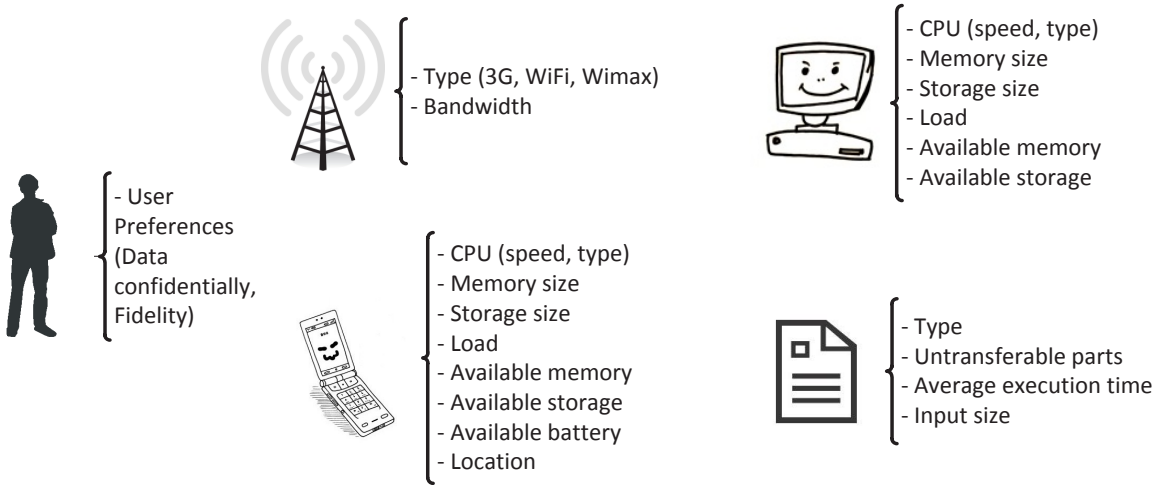

Fig. 2. Important metrics influencing the offloading decision

C. Network Specifications

According to the geographical scope and distance of connected nodes, computer networks are categorized as LAN, MAN or WAN networks [37]. Cyber foraging usually uses the LAN type of networks except in the computational Cloud, which uses the Internet [4], [31]. LAN networks could be wired or wireless, but due to the mobility of mobile device, wireless networks are the only available option for mobile devices. Different types of wireless networks such as 3G, Wi-Fi, and WiMAX have different features and bandwidths. In addition, surrogates can connect via any type of network. Therefore, type, specification, bandwidth, and authentication type of every network is an effective metric for selecting an appropriate communication media between mobile devices and surrogates.

D. Context Specifications

Due to the mobility of mobile device, decisions on offloading of tasks very much depend on the status of devices, surrogates and tasks at the time of decision-making. Available memory and storage spaces, current loads on mobile devices and surrogates, and available battery lifetime of mobile devices are important metrics for taking proper decisions. Current network conditions that can change depending on location and workload, or input size of the application, are another set of effective metrics on offloading decision too.

The users' physiological and mental states, goals, tasks, actions, roles and preferences constitute the contextual properties [53] that affect the offloading decision too. For example, a user can define expected application throughput, confidentiality level of data and allowed latency. It is possible that execution of a program on a surrogate requires the transfer of confidential data to the surrogate. If data is highly confidential, program should not be offloaded to the surrogate at all. Another example is when several translator engines can be used for translations [7] and some engines provide more accuracy and fidelity, albeit consume more resources, energy, and time. User can be free to select the best engine according to the application and the importance of speed and accuracy.

IV. Cyber Foraging Systems

In this section, we survey most credible researches and works on cyber foraging. There are many researches on the cyber foraging area, each focusing on different aspects of cyber foraging to ameliorate resource poverty in mobile devices.

A. Spectra

Spectra [7], [54] is one of the first proposed cyber foraging systems focused on reducing latency and energy consumption. Spectra has added a feature called *self-tuning* to estimate the resources needed to execute an application. It monitors application behavior, measures resource consumption and uses linear regression to model resource demand in terms of application fidelity and input parameters for further prediction of future resource demands. To estimate energy consumption, Spectra does not separate energy rate of various statuses of mobile device (i.e. idle, computing, and communicating). It just monitors energy consumption of two states namely local execution and remote execution. Therefore, when the input data of a task changes, Spectras estimations become inaccurate. Furthermore, to measure the energy consumption of each task, Spectra monitors battery level before and after execution. Therefore, if some tasks execute in parallel, Spectra must throw away the monitored data increasing the required time to reach a good estimation about energy consumption of each task.

Developers must follow most of the cyber foraging steps in Spectra manually significantly changing the code. Before execution, application should call Spectra to determine execution location of each operation. Then application itself is responsible for executing operations according to Spectra's proposed plan. Finally, when the operation is done, application should notify Spectra. All of these commands should be embedded in the application code by the developer. Therefore, there is a need to modify application's code entirely to use Spectra. Furthermore, Spectra is only usable for applications with pre-installed corresponding services on surrogates.

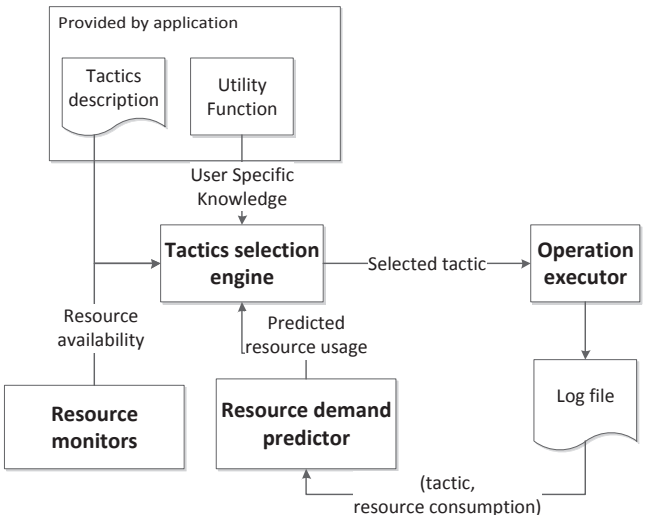

Fig. 3. Architecture of Chroma [56]

B. Chroma

Chroma [6], [55], [56] tries to improve Spectra by reducing the burden on developers. To do so, Chroma uses a new concept called *tactics* that are meaningful ways of application partitioning, specified by the programmer. Tactics differ in application fidelity and the amount of used resources. Upon running of the application, Chroma uses the brute-force method to choose the best or near the best tactic.

On the other hand, user defines a *utility function* for every task that describes the weight and importance of each factor (i.e. CPU speed and energy) in decision making by Chroma. To choose among tactics, Chroma uses a fixed utility function with equal weights for fidelity and latency but ignores battery lifetime. Therefore, a tactic is chosen that maximizes the rate of $fidelity/latency$.

Chroma, like Spectra, uses a history-based approach to predict future resource demands. The proposed mechanism is initialized by offline logging and improves accuracy by online monitoring and machine learning techniques at runtime. Because the determination of resource availability takes time, Spectra and Chroma use probably less up to date and accurate cached results. Spectra and Chroma both assume that the application is installed on the surrogates and there is no need to send the application code. However, this assumption decreases the flexibility and their approach does not work on new surrogates and tasks.

Furthermore, Chroma exploits over-provisioned environments that are full of idle computing resources as follows: (1) it sends a task execution request in parallel to several surrogates and chooses the fastest response; (2) it splits operation data and forwards each part to a different surrogate, wherein the programmer specifies the method of data decomposition and composition; (3) it sends the same task execution request with different fidelities to different surrogates and picks the result with the highest fidelity that satisfies the latency threshold. Figure 3 shows Chroma's architecture.

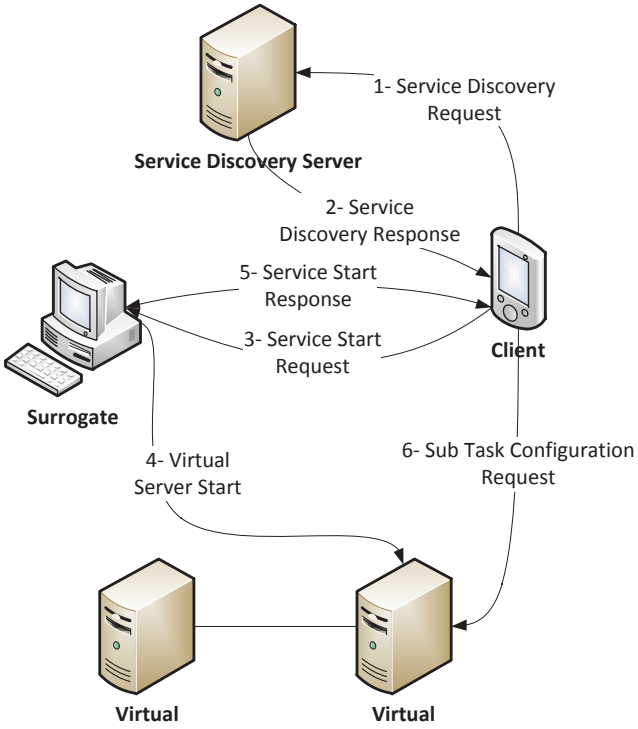

Fig. 4. Control flow of Goyal and Carter's system [42]

C. Goyal and Carter's System

Goyal and Carter (GnC) [42] use the virtual machine technology and present a cyber foraging system that needs Internet connection to increase application performance and decrease energy consumption. The system has a service discovery server that allows all surrogates to register themselves using an XML descriptor file. When a mobile device intends to use the cyber foraging system, it sends a request to the service discovery server and receives the IP address and port number of an appropriate surrogate. Then the mobile device requests a virtual server with specific resource guarantees.

If the surrogate can provide the mobile device's resource demand, it starts a virtual server and sends its IP address to the mobile device. After this step that takes several minutes, the mobile device ships only the URL of the program and a shell script to the surrogate. This shell script is responsible for downloading the real program over the Internet, installing, and running it. Figure 4 shows the control flow of this system.

D. Slingshot

Slingshot [9] is a cyber foraging system based on the virtual machines technology. In this system, the mobile device and surrogates should be connected to the Internet. It assumes that a reliable home server is always accessible via the Internet and if there is no surrogate in the LAN, heavy tasks are offloaded to the home server. It is obvious that higher latency and lower bandwidth in the Internet slows task offloading to the home server than to the nearby surrogates. Figure 5 shows the network topology used by Slingshot.

Upon running of a heavy task, Slingshot sends the task to the home server and all available surrogates. It uses the fastest response, which is probably from one of the surrogates.

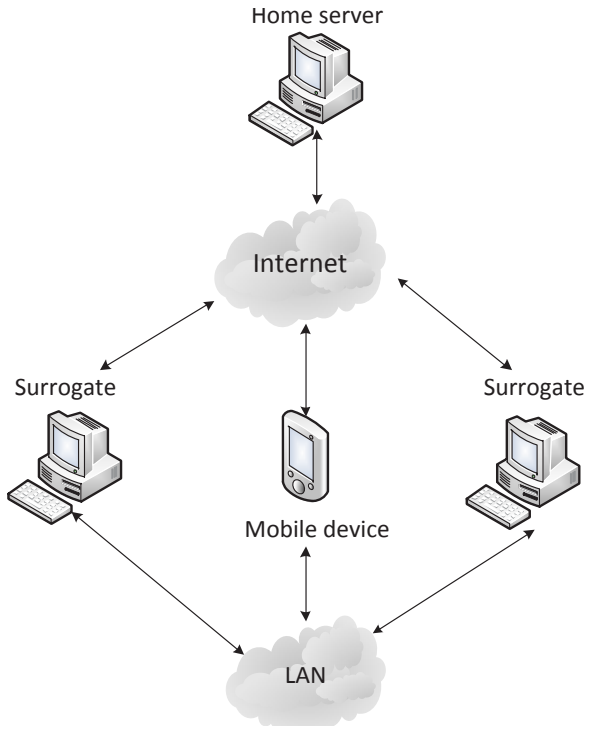

Fig. 5. Network topology used by Slingshot

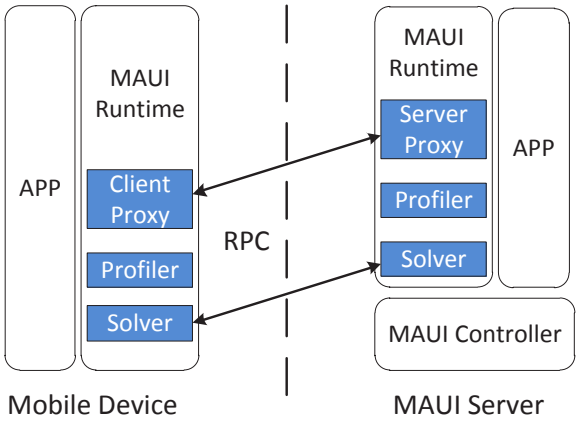

Fig. 6. High-level view of MAUI's architecture [31]

In addition, Slingshot uses the home server's reply to check reliability of the results of available surrogates.

Slingshot presents a good solution for remote execution control, but similar to the GnC's system, does not examine whether task offloading is beneficial in current situation or not.

E. MAUI

MAUI [31] employs fine-grained offloading (i.e. methods of a task) to reduce energy consumption of mobile devices. MAUI supports programs written in managed code environments such as Microsoft .Net CLR and Java. It provides a graph of program's methods and divides them into local and remote groups to execute. Figure 6 shows a high-level view of the MAUI's architecture.

At the mobile device side, MAUI consists of three modules, (1) an interface to the decision unit residing in the MAUI server side, (2) a proxy that is responsible to control a candidate method for offloading, and (3) a profiler that collects information about program's energy and data transfer requirements.

At the server side, MAUI consists of four modules whose proxy and profiler modules are similar to their counterparts in the mobile device. The solver provides the call graph of the program and schedule methods, and the controller is responsible for checking the available requests and to allocate them adequate resources.

MAUI uses online profiling to draw a linear model of energy consumption according to the used processing cycles of each method. In addition, it uses a history-based approach to predict the execution time of tasks. However, it does not consider the effect of input size on execution time of tasks.

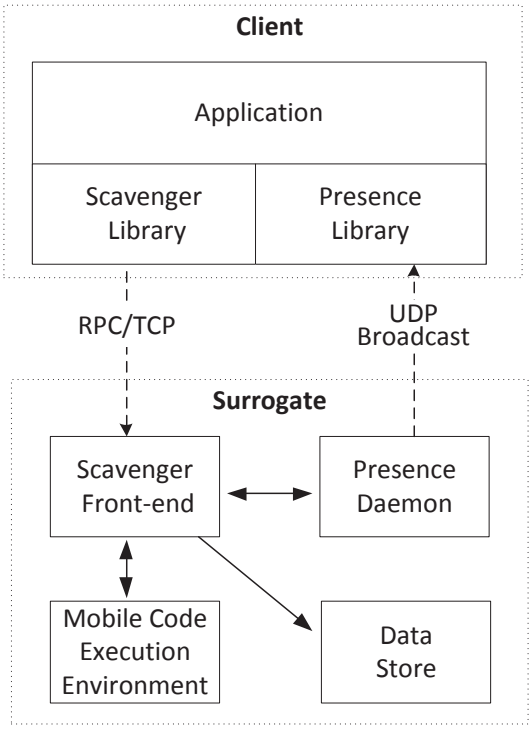

Fig. 7. High-level view of Scavenger's architecture [46]

F. Scavenger

Scavenger [24], [46] is a cyber foraging system that focuses on augmenting CPU power of mobile devices and decrease in the latency of application's response time. Scavenger presents a dual adaptive history-based profiling approach to estimate the execution time of application according to input size and the architecture of execution location.

The Presence Library shown in Figure 7 is responsible for surrogate discovery, and the Scavenger Library schedules tasks and controls remote executions. At the surrogate side, there is the Scavenger front-end that communicates with the mobile device through some RPC entry points.

Scavenger first uses the $Nbench$ benchmark suit to measure a general performance score for mobile device and surrogates and to get a rough estimation of the processing power of each machine. Then, after real execution, it uses online profiling

to improve the estimations of execution time. It considers the effect of input size on execution time by keeping the records of execution time of tasks for different input sizes in separate buckets if their variations are higher than a certain percentage.

Every surrogate periodically sends its processing power ($Peer_{Strength}$) calculated by $Nbench$ and the number of its running tasks ($Peer_{Activity}$) to the mobile device. The surrogate's current processing power is calculated by $Peer_{CurrentStrength} = Peer_{Strength}/(1 + Peer_{Activity})$.

Actually, Scavenger is the only system that considers the CPU utilization effect. However, the $Peer_{CurrentStrength}$ factor cannot suitably represent the effect of CPU utilization and workload because (1) all tasks do not utilize resources equally also this factor ignores the effect of background processes running on the operating system itself, (2) the execution times of different tasks on each architecture are different and the processing power of surrogates must be calculated according to the tasks, which is not considered in this factor, and (3) Scavenger, similar to other mentioned systems, does not measure exactly the estimation factor and other required information before task scheduling. Although such a strategy reduces the scheduling time and decision making's time, but it decreases the precision and accuracy of the decisions.

G. Common Problems

Cyber foraging systems try to reduce the cost of running tasks on mobile devices. Every cyber foraging system must therefore have a good estimation of the costs of local execution and remote execution of a task in order to decide whether to offload the task to a surrogate or not.

All of the above-mentioned works use the online profiling and history-based approach to predict such costs. Actually, they monitor the cost of real execution of every task on every location and use this information to estimate the costs of next runs. Some researches [7], [31], [56] draw a linear model of resource consumption (cost) according to the gathered cyber foraging metrics and some others [46] use the average of previous costs. Although such an approach requires no prior knowledge about the environment and almost all cyber foraging metrics are gathered automatically by the system, it has some shortcomings.

Firstly, since all required information is measured according to previous real executions, cost estimation based on profiles of first runs is not precise. Scavenger uses the $Nbench's$ score to improve first estimations, but this is not effective as we discussed earlier. Secondly, to have a good estimation, some systems [7], [46], [56] keep records of previously measured information about the machine and the execution of tasks. This information can fill up the storage of mobile devices. Scavenger employs cache eviction policy to alleviate this problem. Thirdly, when a task has a wide range of input values, these systems cannot well consider the input data effects on the execution cost estimation.

Although it seems that the current loads of machines affect the cost, none of the mentioned systems considers this effect on estimated cost, except Scavenger.

V. Cyber Foraging Taxonomy

Based on available information on existing cyber foraging systems including those reviewed before in this paper, this section presents our proposed cyber foraging taxonomy. We have used the most important recurring features of cyber foraging systems to categorize and propose this taxonomy. Figure 8 shows the schema of the cyber foraging taxonomy that is discussed in the following subsections. Also Table I shows the place of the discussed cyber foraging systems in each branch of taxonomy.

A. Offloading Type

Offloading can occur at the start-time, referred to as static offloading, or at the run-time, referred to as dynamic offloading [11], [22]. In static offloading, the programmer or a middleware partitions the program prior to execution (at design or installation time). Therefore, at runtime, system knows which parts of program should be offloaded. However, due to the expanded diversity of surrogates and environments, static offloading cannot guarantee to present the best partitioning for all possible situations. Spectra and Chroma are the most important works that do partitioning before program execution.

In contrast, dynamic offloading starts to offload tasks when one of the required resources is insufficient and partitions the program according to the availability of resources at runtime. This approach decides on offloading based on current conditions and is therefore more flexible. It however creates more overheads on the system relating to latency, profiling and run-time decision making that can lead to unnecessary offloading too. Gu *et al.* [14] and Ou *et al.* [47], [52] have used dynamic offloading to improve some of the constraints of mobile device.

To benefit from both static and dynamic advantages, Huerta-Canepa and Lee *et al.* [22] have used a hybrid approach that minimizes the side effects of profiling and waiting time. However, their choice does not work on all patterns and in some cases local execution has better performance than offloading using their proposed scheme. On the other hand, Murarasu and Magedanz [11] have presented a middleware layer, between services and programs, that support reconfiguration of services and programs statically or dynamically and monitors resource consumption and manages the offloading to remote services. Every program executes by a service and there is no need to partition a program.

B. Offloading Granularity

When the application is not available on the surrogate, in addition to request (input data), there is a need to offload related parts of the application to the surrogate too. Referring to offloading granularity strategy, a cyber-foraging approach can offload the partition(s) of a program (i.e. fine-grain) [6], [7], [31], [47], [52], [56], or the whole program (i.e. coarse-grain) [4], [11], [13], [46]

The first strategy is usually relied on programmers to specify how to partition a program and how to adapt the partitions to the changing environment and network conditions.

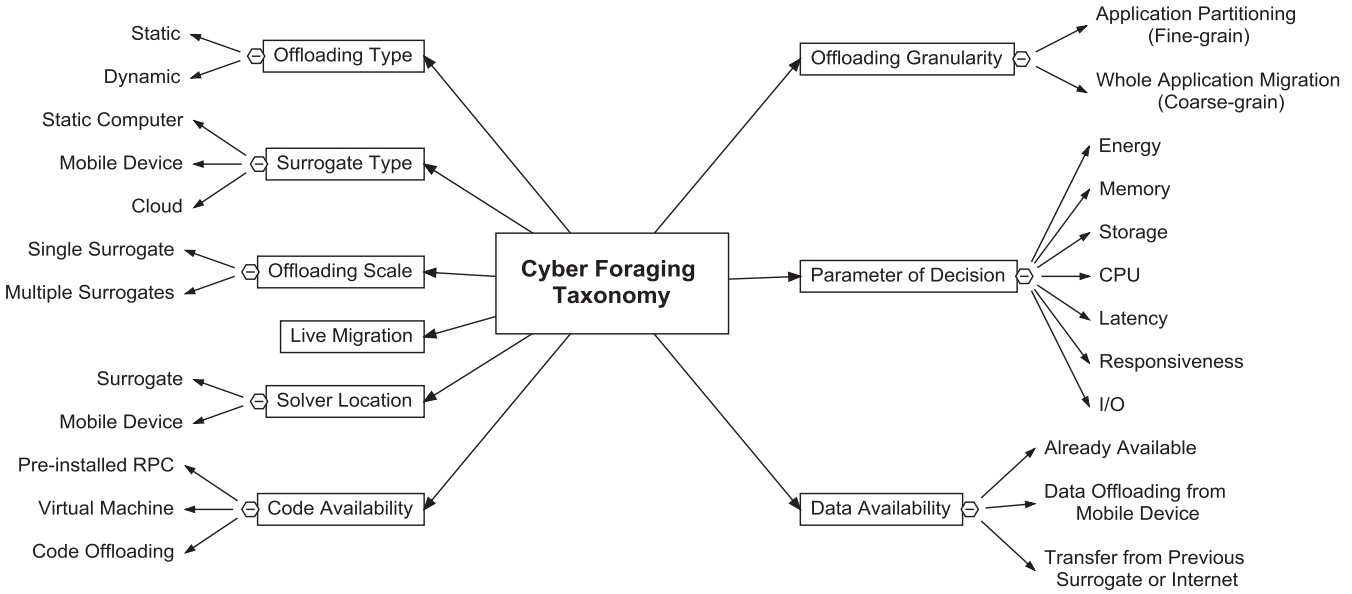

Fig. 8. Taxonomy of cyber foraging approaches

TABLE I
COMPARISON OF CYBER FORAGING SYSTEMS

Metric		Spectra	Chroma	GnC's System	Slingshot	MAUI	Scavenger
Offloading Type		Static	Static	Static	Static	Dynamic	Dynamic
Offloading Granularity		Fine-grained	Fine-grained	Coarse-grained	Coarse-grained	Fine-grained	Coarse-grained
Parameter of Decision		Energy/Latency	Latency	Energy/Performance	Latency	Energy	Latency
Surrogate Type		Stationary/Mobile	Stationary/Mobile	Stationary	Stationary	Stationary/Mobile	Stationary
Parallel Offloading		No	Yes	No	Yes	No	No
Location of the Solver		Mobile device	Mobile device	-	-	Stationary Server	Mobile device
Remote Execution Aspect		Pre-installed RPCs	Pre-installed RPCs	Virtual machines	Virtual machines	Mobile code	Mobile code
Overhead	**Remote Execution**	Low	Low	Low	Low	Low	Low
	Initialization	Low	Low	Low	Low	Low	Low
	Context Gathering	High	High	High	High	High	High
Support of Live Migration		No	No	Yes	Yes	No	No

Partitioning of an application can be done automatically by a cyber foraging system [45] or it can be provided by the programmer [7], [14], [31], [52], [56]. A fine-grain strategy leads to large energy savings as only the parts that benefit from remote executions are offload [31]. Fine-grain granularity is suitable for highly mobile environment(s) [8], wherein mobile devices move in the environment(s) and larger tasks increase the probability of task completion failure due to surrogate disconnection.

In systems with fine grain strategy, such as Spectra, Chroma and MAUI, due to small size of parts and high communication overheads, by taking a local view of each part, offloading does not seem beneficial and the system wrongly decides to execute all parts locally [31]. Therefore, the system should decide for all parts together according to their relations. In Spectra and Chroma, this duty is given to programmers to specify a set of possible partitions along with their suitable execution locations, which are called *execution plans* in Spectra and *tactics* in Chroma.

However, some works, consider the relations between parts to schedule available partitions and to create a graph-based model. In these works, each vertex indicates a part and each edge represents the communication cost between two corresponding vertices. They schedule whole parts together in a manner that parts with more communication stay together in a location and minimize the total cost of task execution. This graph model is usually an NP-Complete problem requiring heuristic solutions. Although, it seems that graph-based scheduling is optimal, but it is shown that scheduling the application's parts as isolated tasks could be more efficient, if the location of input and output data of the tasks are considered [23], [46].

Coarse-grain strategies use the migration of whole virtual machines, processes or requests. This strategy reduces the

programmer's responsibilities because there is no need to modify programs for remote execution and partition it. Except for sending the request for service to surrogates that does not need code migration, in contrast to fine-grain strategies, the whole program state and code must be sent to a remote execution environment. In addition, it does not need to consume resources to solve a graph-based model of partitions. On the other hand, due to the mobility nature of mobile devices, which increases the disconnection possibility and the urgency of connecting to a new surrogate and consequently re-offloading of the whole program to another surrogate, coarse-grain strategies seem to waste a lot of energy and time. Furthermore, they increase the probability of leaving the area in the coverage of surrogates before task completions [23], [41].

C. Parameter of Decision

The goal of cyber foraging is to confront with resource constraints of mobile devices. Therefore, available researches have tried to augment some resources of mobile devices in terms of effective metrics to achieve more efficient application execution. The most important factors that offloading approaches have considered are as follows:

- **Energy Consumption.** One of the most important constraints of mobile devices is energy consumption because mobile device's energy cannot be replenished by itself [15]. Many researches [7], [21], [26], [31], [38] have considered energy consumption as a parameter for offloading decision.
- **Memory and Storage.** Memory intensive applications cannot usually run on mobile devices and they need to be offloaded. Many researches [14], [52] have considered the availability of memory and storage as another effective parameter for offloading decision.
- **Responsiveness.** Offloading decreases execution time when the processing power of mobile devices is considerably lower than static computers. There are many researches [6], [7], [23], [52], [56] that have considered the response time and latency as a major parameter affecting the offloading decision.
- **I/O.** Sometimes, the exploitation of more I/O devices or the improvement of the quality of I/O are the main reason for offloading, e.g. when displaying a movie on a bigger screen, playing music on more powerful speakers, and printing. Some researches [48], [57] have focused on augmenting I/O as an effective parameter on offloading decision.

D. Surrogate Type

We can further categorize cyber foraging approaches by their surrogate type, whether they use static computers or mobile devices. Generally, most cyber foraging approaches use static computers as surrogates [4], [9], [42], [46] though there are some works that use mobile surrogates too [5], [29], [58].

Although powerful stationary computers are considered as suitable surrogates, considerations such as network topology, user preferences, and the absence of idle static computers may guide a cyber foraging system to choose a mobile surrogate instead. In systems such as Spectra, Chroma and MAUI, the mobile device could be potentially a surrogate, too. In current implementation of Scavenger, the mobile device cannot play the role of the surrogate, duo to some code incompatibilities. The same limitation applies to GnC's system as well as to Slingshot because they use the virtualization technology.

Furthermore, in some cases cloud computers, instead of nearby computers play the role of the surrogate. Usually, in contrast to nearby surrogates, using cloud services is not free and also increases response time and energy consumption of mobile devices, due to longer distances and lower bandwidth.

E. Offloading Scale

Offloading scale is another feature that varies in different cyber foraging approaches. In some cases, the cyber foraging system selects only one surrogate from available surrogates to run a task and then waits for the result [7], [21], but in some other works [26], [41], [52] multiple surrogates are used as the offload locations of a task or even the migration of a task between surrogates [9]. A reason is to cope with the mobility nature of mobile devices by increasing the availability of surrogates in the range. In addition, parallel offloading to multiple surrogates are used to increase the fault tolerance [50] and enable the latency control [6], [47], [49], [52], [56].

F. Location of the Solver

Another parameter that branches out of our taxonomy tree of cyber foraging approaches is the location of solver, the unit that is responsible for offloading decisions.

Generally, every mobile device has the role of decision maker and includes a solver, itself [7], [23], [26], [56]. However, in some works [29], [31], the solver is not located on the mobile device. For example, MAUI creates a call graph of application, so if the mobile device itself plays the role of solver, the memory capacity may fill up. In addition, solving this graph-based model takes time and energy and the mobile device's CPU may be 100% utilized. Therefore, there is no choice except leaving the solver to a stationary computer. Although locating the solver away from the mobile device reduces computation cost, it imposes more communication costs to the mobile device.

In GnC's system as well as Slingshot there are no such decision unit and it is supposed that task offloading and remote execution are better than local execution in every situation. However, in Slingshot the mobile device and the home server, in cooperation, control remote execution, while GnC's system uses a stationary computer called the service discovery server to find appropriate surrogates.

G. Remote Execution Aspect

Cyber foraging approaches have different assumptions and strategies on remote execution of offloaded tasks. Cyber foraging approaches can thus be categorized in remote execution respect by their assumptions for code and data availability, and their employed strategies.

1) Code Availability: Kristensen [12], [23] has categorized the existing task execution approaches into three classes: pre-installed RPCs [7], [11], [56], system virtual machines [4], [9], [13], [42], and mobile codes [31], [46]. In the first approach, the task is preinstall on a surrogate and is ready for service. Therefore, the overhead of task execution is small, but it does not work in unknown environments and new surrogates. In contrast, usage of virtual machines increases the flexibility of the system, and surrogates do not need to prepare in advance. However, the overhead of initializing a compatible virtual machine is high. The third approach has the advantage of two previous approaches. If a surrogate does not have the task already, the application code is migrated to the surrogate, compiled, and installed in the surrogate. The overhead of this approach is considerably lower than preparing or bootstrapping the compatible virtual machine, while application code size is far lower than a virtual machine and compile and deployment time is far faster than bootstrapping or creating a virtual machine. In addition, after the first installation of the application, this approach works just like pre-installed RPCs. However, it enforces the availability of the compatible application code.

2) Data Availability: To execute a task, some related information, such as input data, must be available in the execution environment. Assumptions about data availability or strategies for preparing any required data vary among the cyber foraging approaches. The employed assumptions and strategies about data availability fall into three groups. In the first group, data is already available on the surrogate [24]. For example, suppose two tasks A and B where the A's output is the B's input. If a surrogate has executed task A, it has the B's input and it does not need data migration. In the second group, information is transferred from a mobile device to a surrogate [21], [26], [56]. In the third group, necessary information is captured from an old surrogate [9]. This strategy can be extended by fetching the required data from the Internet. It can also be improved by using forecasting methods and context information such as user's location and diary to foresee the next tasks or next available surrogates and prepare to transfer essential information before starting to run the next task.

H. Live Migration Support

Support of live migration means if in the middle of remote task execution, the connection between the mobile device and the corresponding surrogate is disconnected, the cyber foraging system could propose a mechanism to save the current state of process and continue its execution in another location. Among studied research, only those [4], [9], [11], [13], [42] using the virtualization technology provide live migration.

I. System Overhead

Cyber foraging systems have their own overheads including: (1) context gathering and scheduling, (2) initialization of cyber foraging mechanism, and (3) remote execution. The first part is attributed to the overheads of monitoring resource availability, predicting resource demands, assessing costs and making decisions on task execution location. GnC's system and Slingshot are the only cyber foraging systems that do not have this overhead. The overhead of initializing the cyber foraging mechanism refers to the cost of preparing each surrogate to execute each task at the first time. Due to the use of pre-installed RPCs in Spectra and Chroma, and mobile code in MAUI and Scavenger, their initialization overhead is low, in contrast to Slingshot and GnC's system that have high initialization overhead because of using the virtualization technology. The remote execution overhead relates to the overheads of running the task in the surrogate until delivering the result to the mobile device in the next runs, excluding the first and second above-mentioned overheads. This overhead is almost low in all mentioned cyber foraging systems.

VI. Conclusion and Discussion

The ubiquity of mobile devices has allured many users to benefit from their processing abilities too. However, mobile devices are generally more resource constrained than static computers for running complex and high computational applications. Given this background and line of thought, we studied one of the most usable solutions called cyber foraging to augment resource limitations of mobile devices and make the mobile devices amenable for casual use as processing devices too. Cyber foraging is offloading the whole program or a part of it from mobile devices to the nearby static computers (surrogate).

We studied and categorized the effective metrics influencing cyber foraging approaches. In addition, we surveyed six well-known and most credible existing cyber foraging systems and presented our taxonomy of cyber foraging approaches based on recurring features of most common and notable related works on cyber foraging. We categorized our proposed taxonomy based on offloading type that is either static or dynamic, offloading granularity, the resource constraint that is considered by approaches, surrogate type which can be static or mobile, scalability of offloading, location of the solver, availability of code and data on the surrogates, support of live migration, and imposed overheads.

Cyber foraging is a good solution to supplement the resource impoverished of mobile devices, but it has its limitations too. Firstly, some surrogates should be available and eager to share their resources with others via wireless networks. Secondly, there are several security issues in mobile networks [59], [60] and cyber foraging may intensify security and data confidentiality issues. Thirdly, cyber foraging is only applicable to transferable tasks while there are some tasks that are not transferable as discussed in Section II. In addition, cyber foraging may not be beneficial to small tasks due to relatively high communication overhead. Fourthly, although several approaches for offloading applications from mobile devices to static computers have been proposed in recent years, cyber foraging systems have a long way to go to provide all mentioned steps including surrogate discovery, context gathering, partitioning, scheduling, and remote execution control, and become deployable in real world. This becomes more on sight if a strong development support is provided in a way that even a novice programmer becomes capable to enable cyber foraging support.

According to our survey and the issues raised in this paper, three main areas in cyber foraging grant future research. First,

to make a good decision about the execution location of each task, it is important to have a good estimation of execution cost on every location before real execution. Therefore, there is a need to estimate the cost for local and remote executions in a more precise and reliable manner, considering the effect of input size of the task and the current load of machines (i.e. the mobile device and available surrogates). As stated before, available researches usually use history-based approaches to predict the cost and cannot consider well the current load and input data effects on the execution cost estimation. It seems that the composition of these techniques and some pre-knowledge about application structure and specifications can improve the estimations.

The second challenge is context gathering. Current available researches of cyber foraging gather context information and cyber foraging metrics periodically, instead of just before execution of task that contains the real and precise context information. Periodic profiling removes the overhead of context gathering just before executing the task and increases the performance. However, it has two shortcomings: (1) it decreases the accuracy of decisions that are based on historical data and (2) if there is no demand to execute a task on the mobile device for a long time, periodic context gathering is useless and just burdens the system with unnecessary overhead. A solution to this problem could be the mixture of static and dynamic context gathering. All context information is gathered at first time and only variable information is updated just before real execution. This solution needs to recognize variable information and find a way to gather them quickly and precisely.

Offloading granularity is the third challenge that requires further research. Coarse-grain offloading reduces the burden off the programmers' shoulders and reduces the overhead of partitioning and scheduling. However, it is less akin to newly envisaged real world applications running in highly mobile environments. New approaches for coarse-grain and fine-grain orchestrated offloading according to the nature and specifications of the residing environment of mobile devices and surrogates are required.

REFERENCES

[1] UN, "The global partnership for development at a critical juncture," United Nations, MDG GAP Task Force Report, 2010.

[2] D. Chalmers and M. Sloman, "A survey of quality of service in mobile computing environments," *IEEE Commun. Surveys Tutorials*, vol. 2, no. 2, pp. 2–10, 1999.

[3] M. Perry, K. O'hara, A. Sellen, B. Brown, and R. Harper, "Dealing with mobility: Understanding access anytime, anywhere," *ACM Trans. Computer-Human Interaction (TOCHI)*, vol. 8, no. 4, pp. 323–347, 2001.

[4] M. Satyanarayanan, P. Bahl, R. Cceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[5] J. Oh, S. Lee, and E. Lee, "An adaptive mobile system using mobile grid computing in wireless network," in *International Conference on Computational Science and Its Applications (ICCSA 2006)*, Glasgow,UK, 2006, pp. 49–57.

[6] R. K. Balan, D. 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.

[7] J. Flinn, S. Park, and M. Satyanarayanan, "Balancing performance, energy, and quality in pervasive computing," in *22nd International Conference on Distributed Computing Systems (ICDCS02)*, Vienna, Austria, 2002, pp. 217–226.

[8] M. D. Kristensen, "Enabling cyber foraging for mobile devices," in *5th MiNEMA Workshop*, Magdeburg, Germany, 2007, pp. 32–36.

[9] Y. Y. Su and J. Flinn, "Slingshot: Deploying stateful services in wireless hotspots," in *3rd International Conference on Mobile Systems, Applications, and Services*, New York, NY, USA, 2005, pp. 79–92.

[10] M. D. Kristensen and N. O. Bouvin, "Developing cyber foraging applications for portable devices," in *2nd IEEE International Interdisciplinary Conference on Portable Information Devices*, Garmisch-Partenkirchen, Germany, 2008, pp. 1–6.

[11] A. F. Murarasu and T. Magedanz, "Mobile middleware solution for automatic reconfiguration of applications," in *6th International IEEE Conference on Information Technology*, Las Vegas, USA, 2009, pp. 1049–1055.

[12] J. Porras, O. Riva, and M. D. Kristensen, *Dynamic Resource Management and Cyber Foraging*. Berlin And Heidelberg: Springer, 2009, ch. 16, pp. 349–368.

[13] B. Chun and P. Maniatis, "Augmented smartphone applications through clone cloud execution," in *12th Workshop on Hot Topics in Operating Systems (HotOS)*, Monte Verita, Switzerland, 2009.

[14] X. Gu, A. Messer, I. Greenbergx, D. Milojicic, and K. Nahrstedt, "Adaptive offloading for pervasive computing," *IEEE Pervasive Computing Mag.*, vol. 3, no. 3, pp. 66–73, 2004.

[15] M. Satyanarayanan, "Avoiding dead batteries," *IEEE Pervasive Computing*, vol. 4, no. 1, pp. 2–3, 2005.

[16] R. K. Balan, "Powerful change part 2: Reducing the power demands of mobile devices," *IEEE Pervasive Computing*, vol. 3, no. 2, pp. 71–73, 2004.

[17] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker, "Agile application-aware adaptation for mobility," in *16th ACM Symp. Operating Systems Principles (SOSP)*, Saint-Malo, 1997, pp. 276–287.

[18] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low power embedded operating systems," in *18th Symp. Operating System Principles (SOSP)*, Banff, Canada, 2001, pp. 89–102.

[19] M. Satyanarayanan, "Pervasive computing: Vision and challenges," *IEEE Personal Commun.*, vol. 8, no. 4, pp. 10–17, 2001.

[20] T. E. Starner, "Powerful change part 1: Batteries and possible alternatives for the mobile market," *IEEE Pervasive Computing*, vol. 2, no. 4, pp. 86–88, 2003.

[21] 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, 1998.

[22] G. Huerta-Canepa and D. Lee, "An adaptable application offloading scheme based on application behavior," in *22nd International Conference on Advanced Information Networking and Applications Workshops (AINAW2008)*, Gino-wan City, Okinawa, Japan, 2008, pp. 387–392.

[23] M. D. Kristensen, "Empowering mobile devices through cyber foraging:the development of scavenger, an open mobile cyber foraging system," Ph.D. dissertation, Aarhus University, 2010.

[24] M. D. Kristensen, "Scavenger: Transparent development of efficient cyber foraging applications," in *IEEE International Conference on Pervasive Computing and Communications (PerCom)*, Mannheim, Germany, 2010, pp. 217–226.

[25] S. Kalasapur and M. Kumar, "Resource adaptive hierarchical organization in pervasive environments," in *1st International Conference on Communication Systems and Networks*, Bangalore, 2009, pp. 9–16.

[26] E. Park, H. Shin, and S. J. Kim, "Selective grid access for energy-aware mobile computing," *Lecture Notes in Computer Science(LNCS)*, vol. 4611, pp. 798–807, 2007.

[27] L. Chunlin and L. Layuan, "Energy constrained resource allocation optimization for mobile grids," *Journal of Parallel and Distributed Computing*, vol. 70, no. 3, pp. 245–258, 2010.

[28] O. Storz, A. Friday, and N. Davies, "Towards 'ubiquitous' ubiquitous computing: an alliance with the grid," in *1st Workshop on System Support for Ubiquitous Computing Workshop (Ubisys 2003) in association with 5th International Conference on Ubiquitous Computing*, Seattle, 2003.

[29] Y. Begum and M. Mohamed, "A DHT-based process migration policy for mobile clusters," in *7th International Conference on Information Technology*, Las Vegas, 2010, pp. 934–938.

[30] B. Chun and P. Maniatis, "Dynamically partitioning applications between weak devices and clouds," in *1st ACM Workshop on Mobile Cloud Computing and Services (MCS 2010)*, San Francisco, 2010, pp. 1–5.

[31] E. Cuervo, A. Balasubramanian, D.-k. 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.

[32] K. Kumar and Y. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *IEEE Computer*, vol. 43, no. 4, pp. 51–56, 2010.

[33] R. Buyya, S. Yeo, Chee, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[34] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: a computation offloading framework for smartphones," in *3rd International Conference on Mobile Computing, Applications, and Services (MobiCASE)*, Santa Clara, CA, USA, 2010.

[35] R. Kemp, et al., "The smartphone and the cloud: Power to the user," in *International Workshop on Mobile Computing and Clouds (MobiCloud)*, Santa Clara, CA, USA, 2010.

[36] B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "CloneCloud: elastic execution between mobile device and cloud," in *6th conference on Computer Systems (EuroSys)*, Salzburg, Austria, 2011.

[37] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Unleashing the power of mobile cloud computing using ThinkAir," 2011.

[38] R. Kemp, N. Palmer, T. Kielmann, F. Seinstra, N. Drost, J. Maassen, and H. Bal, "eyeDentify: Multimedia cyber foraging from a smartphone," in *IEEE International Symposium on Multimedia (ISM2009)*, San Diego, 2009, pp. 392–399.

[39] A. P. Miettinen and J. K. Nurminen, "Energy efficiency of mobile clients in cloud computing," in *2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, Boston, MA, USA, 2010.

[40] M. Nkosi and F. Mekuria, "Cloud computing for enhanced mobile health applications," in *IEEE 2nd International Conference on Cloud Computing Technology and Science (CloudCom)*, Indianapolis, IN, USA, 2010.

[41] M. D. Kristensen, "Execution plans for cyber foraging," in *1st Workshop on Mobile Middleware: Embracing the Personal Communication Device*, Leuven, Belgium, 2008, pp. 87–92.

[42] S. Goyal and J. Carter, "A lightweight secure cyber foraging infrastructure for resource-constrained devices," in *6th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA '04)*, English Lake District, UK, 2004, pp. 186–195.

[43] M. D. Kristensen, "Scavenger - mobile remote execution," University of Aarhus, Technical Report DAIMI PB-587, 2008.

[44] C. N. Ververidis and G. C. Polyzos, "Service discovery for mobile ad hoc networks: A survey of issues and techniques," *IEEE Commun. Surveys Tutorials*, vol. 10, no. 3, pp. 30–45, 2008.

[45] G. C. Hunt and M. L. Scott, "The coign automatic distributed partitioning system," in *3rd Symposium on Operating Systems Design and Implementation (OSDI'99)*, New Orleans, 1999, pp. 187–200.

[46] 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, 2010.

[47] 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 (PERCOM06)*, Pisa, Italy, 2006, pp. 116–125.

[48] X. Song and U. Ramachandran, "MobiGo: A middleware for seamless mobility," in *13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'2007)*, Daegu, 2007, pp. 249–256.

[49] R. K. Balan, "Simplifying cyber foraging," PhD Thesis, Carnegie Mellon University, 2006.

[50] X. Zhang, S. Jeong, A. Kunjithapatham, and S. Gibbs, "Towards an elastic application model for augmenting computing capabilities of mobile platforms," in *3rd International ICST Conference on Mobile Wireless Middleware, Operating Systems, and Applications (MobileWare)*, Chicago, USA, 2010, pp. 161–174.

[51] J. Zhang and R. J. Figueiredo, "Application classication through monitoring and learning of resource consumption patterns," in *20th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Rhodes Island, 2006.

[52] S. Ou, K. Yang, and Q. Zhang, "An efficient runtime offloading approach for pervasive services," in *IEEE Wireless Communications and Networking Conference (WCNC2006)*, Las Vegas, 2006, pp. 2229–2234.

[53] J. Krogstie, "Requirement engineering for mobile information systems," in *7th International Workshop on Requirements Engineering*, Interlaken, Switzerland, 2001.

[54] 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.

[55] R. K. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, and H. Yang, "The case for cybef foraging," in *10th Workshop on ACM SIGOPS European Workshop: beyond the PC*, New York, NY, USA, 2002.

[56] 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.

[57] X. Song, "Seamless mobility in ubiquitous computing environments," PhD Thesis, Georgia Institute of Technology, 2008.

[58] G. Huerta-Canepa and D. Lee, "A virtual cloud computing provider for mobile devices," in *1st ACM Workshop on Mobile Cloud Computing and Services: Social Networks and Beyond(MCS10)*, San Francisco, 2010, pp. 1–5.

[59] D. Djenouri, L. Khelladi, and N. Badache, "A survey of security issues in mobile ad hoc and sensor networks," *IEEE Commun. Surveys Tutorials*, vol. 7, no. 4, pp. 2–28, 2005.

[60] M. N. Lima, A. L. d. Santos, and G. Pujolle, "A survey of survivability in mobile ad hoc networks," *IEEE Commun. Surveys Tutorials*, vol. 11, no. 1, pp. 66–77, 2009.

Mohsen Sharifi is an Associate Professor of Software Engineering currently with the School of Computer Engineering of Iran University of Science and Technology. He directs the distributed systems research group and laboratory in the school. His main interest is in the development of kernel level and middleware level distributed system software for use in mission critical applications requiring dependable high performance computing capabilities. He received his B.Sc., M.Sc. and Ph.D. in Computer Science from the University of Manchester in United Kingdom.

Somayeh Kafaie received her B.Sc. in Computer Engineering (Software) from Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran and her M.Sc. in Computer Engineering (Software) from Iran University of Science and Technology, in 2007 and 2011, respectively. Her research interests include energy-efficient systems and distributed computing especially pervasive and mobile computing.

Omid Kashefi received his B.Sc. and M.Sc. in Computer Engineering (Software) from the School of Computer Engineering of Iran University of Science and Technology in 2006 and 2009, respectively. His main research interests include distributed systems, operating systems, and virtualization.