7-2017

Cyber foraging: Fifteen years later

Rajesh Krishna BALAN
Singapore Management University, rajesh@smu.edu.sg

Jason FLINN

DOI: https://doi.org/10.1109/MPRV.2017.2940972

Published in IEEE Pervasive Computing, 2017 July, Volume 16, Issue 3, Pages 24-20
https://doi.org/10.1109/MPRV.2017.2940972

Cyber Foraging: Fifteen Years Later

Revisiting Mahadev Satyanarayanan's original vision of cyber foraging, the authors reflect on the last 15 years of research progress and discuss accomplishments and remaining challenges. They also consider compelling application scenarios required to make cyber foraging a widely deployed technology.

Rajesh Krishna Balan
Singapore Management University

Jason Flinn
University of Michigan

The term *cyber foraging* was first coined by Mahadev Satyanarayanan (Satya) in his 2001 *IEEE Pervasive Communications* article, "Pervasive Computing: Visions and Challenges."[1] The term captures the vision of computers that can "live off the land" by acquiring, from their immediate environment, the resources needed to perform various tasks. In Satya's vision, mobile devices offload computation to servers called *surrogates*, which are located in close proximity to the mobile device. Surrogates are located in "public spaces such as airport lounges and coffee shops... much as table lamps are today."[1] To facilitate this vision, Satya stated that it was necessary to develop systems to partition applications between local and remote resources and to manage and secure the surrogates, data, and devices needed for offloading computation.

Satya further described cyber foraging with the following scenario:

> *When a mobile computer enters a neighborhood, it first detects the presence of potential surrogates and negotiates their use. Communication with a surrogate is via short-range wireless peer-to-peer technology, with the surrogate serving as the mobile computer's networking gateway to the Internet. When an intensive computation accessing a large volume of data has to be performed, the mobile computer ships the computation to the surrogate; the latter may cache data from the Internet on its local disk in performing the computation. Alternatively, the surrogate may have staged data ahead of time in anticipation of the user's arrival in the neighborhood. In that case, the surrogate may perform computations on behalf of the mobile computer or merely service its cache misses with low latency by avoiding Internet delays. When the mobile computer leaves the neighborhood, its surrogate bindings are broken, and any data staged or cached on its behalf are discarded.*

At the heart of this vision are the surrogates providing resources that mobile devices can leverage to perform computational tasks. This leveraging, using various offloading techniques, can be performed to reduce application latency, improve application performance or quality, and reduce the energy consumed by the mobile device. This vision has inspired a tremendous amount of research in the mobile computing community, leading to the development of numerous cyber foraging prototypes. As of yet, however, academic research has not spilled over into industry; there is currently a noticeable lack of commercial deployments that use cyber-foraging technology.

In this retrospective article, we look back at the last 15 years of research in cyber foraging and ask the following questions:

- Have we overcome the technical challenges required to achieve seamless cyber foraging?
- What challenges still remain, and how do they hamper the adoption of cyber foraging?
- What does the future look like for cyber foraging research and the deployment of cyber-foraging systems?

To answer these questions, we summarize relevant research and technical accomplishments in partitioning and surrogate management and discuss the difficulties of finding the killer app for cyber foraging and tackling the challenges of surrogate deployment. We also identify a new class of video processing applications, driven by the emergence of portable augmented and virtual reality (VR) displays.

Accomplishments

The major accomplishments on the path to achieving cyber foraging fall into two broad categories: learning how to partition applications between local and remote resources, and understanding how to better manage and secure offloaded computation.

Partitioning

Simply stated, the partitioning decision answers the following question: Given a specific application state and a specific computational environment, which portions of the application should run on the mobile computer, and which should run on remote infrastructure?[2]

In addressing this issue, there are many possible answers. However, the consensus that has emerged in the field is that the two most important objectives of the partitioning decision are maximizing application performance and minimizing energy usage on the mobile computer (thereby preserving precious battery lifetime). Performance can be expressed in various metrics, depending on the application requirements. For example, early cyber-foraging systems (such as Spectra[3]) that execute discrete tasks seek to minimize task completion time. Later systems (such as Odessa[4]) that target stream-based computation try to improve makespan and throughput. Some systems let applications alter the computation to produce results of varying fidelity; in such systems, maximizing fidelity is also important.[3,5]

Balancing multiple objectives is nontrivial, because each goal is expressed in different units (as time for performance, in Joules for energy, and as an application-specific measure of quality for fidelity). Two approaches have emerged for dealing with this issue. Some systems explicitly balance competing goals by first defining a static or dynamic utility function that converts different metrics into a single cost function and then minimizing that cost function.[3,5] Others seek to optimize a single goal subject to constraints on the other goals.[6]

Researchers have explored many different granularities for partitioning applications and executing some functionality remotely. Early cyber-foraging systems relied on developers to explicitly specify large components in their code, which could be executed on remote computers.[3,5] More recent systems (such as CloneCloud[7]) have leveraged reflection in managed language runtimes to automatically partition applications at method boundaries. Some systems (such as MAUI[6]) take a hybrid approach; developers annotate methods that can be executed remotely, and MAUI uses reflection to determine at runtime which, if any, of these should execute remotely. Comet uses thread-level partitioning by migrating threads between the mobile computer and a remote server such as a surrogate.[8]

Despite this variation in granularity, researchers have tended to agree on the properties that make a code component well-suited for remote execution. First of all, the component should not produce external output, because that output can't be reliably reproduced on the mobile computer if the surrogate fails. Consider an offloaded computation that generates a random key, encrypts network communication with a cloud server using that key, and then fails. The mobile computer can't seamlessly continue communicating with the cloud server, because it lacks the random key needed to encrypt and decrypt communication.

To be suitable for offloading, a code component should have a substantial amount of computation yet small inputs and outputs.

Second, the ratio of computation to communication should generally be high. Offloading substantial computation to a surrogate improves performance and reduces mobile computer energy usage. However, the mobile computer must send the computation's inputs to the surrogate and must receive the results of the computation; both require network communication that comes at a performance and energy cost. Therefore, to be suitable for offloading, a code component should have a substantial amount of computation yet small inputs and outputs.

Finally, the component should have few or no dependencies on external input and system services on the mobile computer, because resolving such dependencies would require additional network communication. One challenge for systems such as CloneCloud and Comet, which automatically identify components to offload, is that there

might be only a few large components that meet these criteria in typical mobile applications. If developers must modify their applications to create such components, then the additional overhead of annotating the components they create could be relatively low.

To decide whether to offload a component to a surrogate, cyber-foraging systems typically rely on observations of historical behavior. This helps the systems determine the component's computational demand and the communication overhead required to offload the component. Many systems also observe the environment to determine the resource supply, such as network latency and bandwidth, CPU load on the mobile computer and surrogate, and remaining battery energy. The combination of these two factors—supply and demand—yields predictions for how well local or remote execution would satisfy the goals of the cyber-foraging system; predicting, for example, how offloading might impact application performance and mobile computer energy usage.

More recent work in cyber foraging has recognized that such predictions can often be unreliable and lead to nonideal behavior. For example, applications might exhibit considerable variance in CPU demand or data size as a result of variation in program input or user behavior. Mobile networks can also exhibit substantial changes in latency, bandwidth, and reliability due to movement, radio interference, and load variation. If the cyber-foraging system mispredicts either supply or demand, its decision about whether to execute a component locally or remotely might be incorrect. Even worse, a mobile network or surrogate might fail, leading to the inability to complete a remote computation or receive its results.

One strategy to address such behavior, used in systems such as MAUI,[6] is to time out if a remote computation takes too long to complete and fail over to local execution. Chroma[5] and Slingshot[9] mitigate uncertainty in server execution and network communication time by potentially executing a component on more than one remote server and using the fastest response to continue the application's execution. Such an approach reduces both average execution time and tail latency when response times are both uncertain and independent of one another. Similarly, the cyber-foraging system can simultaneously execute a component on both the mobile computer and a surrogate[10,11] to improve performance.

Progress in surrogate management has been made along the dimensions of isolation of remotely hosted computation, state management and provisioning, and surrogate location.

Management

The cyber-foraging research community has also made considerable progress in surrogate management. In the original "Vision and Challenges" paper,[1] Satya defined surrogates to be remote computers that might temporarily help a mobile computer (by hosting offloaded computation, for example). Progress in surrogate management has been made along the dimensions of isolation of remotely hosted computation, state management and provisioning, and surrogate location.

Isolation. A surrogate can host offloaded computation from several mobile computers, so it is important to isolate each hosted computation from other computations and from variation in the execution environment between the surrogate and mobile computer. For correct behavior, surrogates must uphold the *result equivalence* property:

> *The observable results of an operation that executes remotely on a surrogate should be indistinguishable from results that could have been produced by the same operation if it had executed on a mobile computer.*[2]

Upholding the result equivalence property allows transparent offloading of computation from the mobile device to the surrogate. It is typically provided by executing the offloaded computation within an isolated sandbox such that the external inputs (results of system calls, invocations of middleware services, or input from devices and the user) to the sandbox on the remote surrogate are the same as those that would have been received on the mobile computer.

Cyber-foraging researchers have explored three techniques to provide isolation. Early cyber-foraging systems used process-level isolation provided by the operating system.[3,12] Such isolation is lightweight, leading to excellent performance. However, processes have many external dependencies—including the operating system, the dynamic libraries, and services running on the host computer. If a hosted computation interacts with any of these dependencies, then the cyber-foraging system must ensure that the operating systems, libraries, and external services on the surrogate are compatible with those on the mobile computer in order to uphold the result equivalence property. In practice, this proved to be quite difficult due to the heterogeneity of mobile computers.

Subsequent cyber-foraging systems used hardware virtual machines (VMs) for isolation.[9,13] Hardware virtualization provides strong isolation because each VM contains its own version of the operating system, libraries, and

middleware services. However, the performance cost of this isolation can be steep because of the large amount of state needed to encapsulate all of those entities; the state must be saved, transmitted over the network, and restored to execute a computation remotely.

Application virtualization in managed runtimes, such as C#[6] or Dalvik,[7] represent a middle ground. Theoretically, the language-level virtualization can provide strong isolation with a much lower cost due to reducing the state size within each VM. However, modern applications running on mobile computers execute a great deal of native code and often invoke middleware or system services during execution. MAUI handled these external dependencies by having developers annotate which methods could be executed remotely and which could not (because they rely on dependencies that might not exist or that might have different states on the surrogate, for example). Tango found that handling such dependencies was a considerable source of complexity when supporting cyber foraging for Android platforms.[10]

Thus, although cyber-foraging researchers have explored many different options for isolation, no choice clearly dominates across all dimensions. Methods such as hardware virtualization that provide strong isolation also incur considerable performance overhead; methods such as process-level isolation that provide low overhead also have weak isolation. Resolving this tradeoff has proven difficult.

State management and provisioning. Surrogates encapsulate the application-specific state needed to perform an offloaded computation within one of the isolation mechanisms just described. At a minimum, this state consists of the inputs needed for the computation and the executable code required to perform the computation. Depending on the isolation mechanism, the state might also comprise libraries and services used by the computation—or even an entire operating system in the case of hardware virtualization.

Researchers have developed a variety of methods to synchronize the state between mobile computers and surrogates. One method used by MAUI[6] and CloneCloud[7] directly transfers the inputs required by a computation to the surrogate before beginning the offloaded computation. These systems rely on reflection within the language runtime to discover which objects might be accessed by the computation; however, the set of objects actually accessed might be potentially smaller than the set that could be accessed, leading to lower performance and increased network usage.

Comet uses distributed shared memory to only transfer the state that is actually accessed by both the mobile computer and remote surrogate.[8] This potentially reduces the bytes transferred, but such state must be transferred on demand, which can incur performance overhead due to network latency. Tango deterministically reproduces the same state on the mobile computer and surrogate by running identical executions on the two platforms.[10] This leads to a further reduction in bytes transferred, but incurs the overhead of replicating computation.

Systems such as Spectra supplement direct transfer using an external distributed storage system to transfer a large state such as dynamic libraries and data files.[3] Because many such systems provide weak data consistency, researchers have often found it necessary to provide them with explicit synchronization hints to ensure that needed inputs are transferred before computation begins.

When the state is very large—for example, with hardware virtualization—researchers have exploited commonality in state across offloaded computations to decrease the amount of state that is transferred for each individual computation. Early work on cloudlets used *VM overlays*[14] to transfer only the difference between a customized state and a more generic reference state. Kiryong Ha and his colleagues showed that transferring a VM overlay and instantiating it over the image of a base VM image could provision a custom VM in as little as 10 seconds.[15] Alternatively, systems such as Slingshot[13] have used content-addressable storage to find common chunks among multiple VM states, transferring only those chunks that are unknown to the receiver. Ha and his colleagues combine these two techniques by deduplicating chunks within each VM overlay.

Surrogate location. The emergence of cloud computing has led to the centralization of compute services within the datacenter. Most mobile applications that rely on remote computation currently execute that computation in a cloud datacenter. It is therefore worth considering whether the original vision of surrogates located near mobile computers is still valuable.

The developers of MAUI investigated how network latency impacted the performance of applications with offloaded computation.[6] For some applications, such as face recognition or a video game, the difference between local network latencies and mobile-to-cloud latencies resulted in changes in application performance of up to 50

Although cyber-foraging researchers have explored many different options for isolation, no choice clearly dominates across all dimensions.

percent. Furthermore, mobile computer energy usage was often greater than if the computation had not been offloaded at all. Using the MAUI approach, other applications, such as real-time processing of captured video, might require substantial amounts of data to be sent from the mobile computer to the surrogate. The bandwidth available over a local Wi-Fi network might greatly exceed that available over the backhaul link; bandwidth to surrogates within cellular infrastructure might be greater than the bandwidth available to a datacenter. Thus, when offloading computation from user-facing, interactive applications, there can be considerable benefit to using a nearby surrogate rather than one in the cloud.

The cloudlet vision[14] builds on this observation. While cloudlets have many aspects, an essential feature for cyber foraging is that they provide low-latency, high-bandwidth, one-hop wireless network connectivity to mobile computers. Cloudlets can be deployed with Wi-Fi base stations or within cellular network infrastructure. Thus, they represent an updated vision for surrogates that are located nearby mobile computers.

Industry Solutions

Some parts of the cyber-foraging vision have already become standard industry practices. For example, many services are offered through a client-server model in which the client sends inputs to a server that processes the request on behalf of the client and returns the output. For example, speech recognition, language translation, and real-time navigation all work very well using a client-server model, because they only require soft real-time guarantees (a few hundred milliseconds, because a human is processing the answers) and have reasonably small inputs and outputs. However, the client-server model does not work well for services that require hard real-time guarantees, such as real-time multiplayer gaming and real-time vision analytics, or where the application inputs and outputs are large relative to the available bandwidth (for example, real-time high resolution video processing).

The client-server model does not work well for services that require hard real-time guarantees, such as real-time multiplayer gaming and real-time vision analytics.

Cloud computing has addressed many important issues pertaining to isolation and management. Researchers have found cloud solutions, such as VMs and containers, translate well to cyber foraging. However, surrogates present additional challenges compared to servers in cloud datacenters. The lack of physical security makes securing computation and data more challenging on surrogates, and the lack of easy access makes maintenance more difficult to perform. Surrogates have less colocated resources than datacenter servers, so handling large datasets requires careful data partitioning and caching.

There has also been a large effort to provide content distribution networks (CDNs) that are located in close proximity to as many clients as possible. These CDNs are provided by companies such as Netflix, Amazon, Google, Akamai, and Microsoft, with the goal of making the download of stored content (such as webpages and video data) as fast as possible. These CDNs, however, usually allow only fast downloading of stored content and do not let clients upload and share content with other clients. CDNs are not currently a solution for clients that want to offload client-specific computation.

Remaining Challenges

Despite the accomplishments described earlier, there are at least two major reasons why cyber foraging has not yet proven to be commercially viable: a compelling killer application has not emerged, and deploying and maintaining surrogates is challenging.

Missing a Compelling Application

Many applications for cyber foraging have been proposed over the years, such as language translation and speech recognition,[5,16,17] face recognition,[5,6,16] and graphics processing.[5,6,16,18] Researchers hypothesized that the latency to run these applications on a remote server would be higher than a user could tolerate and that running these applications on a mobile phone would be computationally infeasible—creating "perfect" conditions for cyber foraging.

However, two developments have challenged this hypothesis. First, for many of these applications—such as those for language translation, face detection, and speech recognition—the required dataset needed for them to run is large and proprietary. In addition, newer algorithms, such as deep-learning-based approaches, have greatly increased the accuracy of these solutions when using a large dataset. These factors naturally make these services amenable as web services, where the full dataset is used for every request (to improve accuracy) while caching, pipelining, and other techniques are used to reduce latency to acceptable levels.

Second, the exponential improvement in mobile processing capabilities has allowed smaller versions of these applications, which use small local datasets to run in real time on modern phones. For example, modern cell phones have local speech recognizers

that can accurately detect a subset of words in real time (such as "OK Google"). The GPUs of modern phones are also powerful enough to process many graphics tasks locally with acceptable performance and latency.

To date, there have been no compelling mass-market applications that require low latencies that cannot be achieved as a web service and that also are too computationally or energy intensive for modern smart phones to run locally. This might be a "chicken or the egg" problem: the lack of cyber foraging infrastructure could potentially be hindering the development of such applications. Later, we discuss one emerging class of applications that could prove to be the compelling application that cyber foraging needs.

The Challenge of Server Setup and Maintenance

The second hurdle for cyber-foraging adoption is the challenge of server setup and maintenance. For example, moving one of the services offered as a web service, such as language translation, to a cyber-foraging deployment would require

- ensuring adequately provisioned surrogates are located near a majority of mobile users,
- offering user credentials and security primitives to easily authenticate mobile users to surrogates and vice versa,
- transferring all datasets required for application use to the surrogates,
- protecting proprietary datasets from leaking information to mobile users or other applications running on potentially shared surrogates,
- running the applications requested by the mobile users, and
- migrating the user state (if required) between surrogates or the cloud to maintain network proximity as users themselves move.

Looking at the requirements a little deeper, we see that the initial step of providing local surrogates for mobile usage is already a substantial barrier. In particular, who should provide these surrogates? If the application provider must contribute surrogates, the cost of providing local computation will be prohibitively high. If surrogates are provided by users or by third-party infrastructure providers (such as cellular companies and ISPs), the challenge becomes providing adequate security and privacy so that application providers feel comfortable moving their proprietary code and datasets onto surrogates that they don't themselves control. Finally, convincing users to use third-party surrogates that are not the "authoritative" web service might require new authentication, security, and privacy mechanisms to be developed, maintained, and explained (to the users).

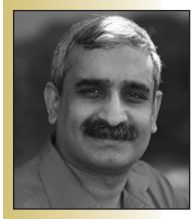
All of these challenges could be addressed if there were sufficiently compelling use cases that require cyber foraging. However, the lack of such use cases, coupled with high setup costs, creates substantial barriers for commercial cyber-foraging deployment.

Looking Forward

Recently, a new class of applications requiring real-time video processing is emerging as a potential candidate for cyber foraging. This class of application comes in two main forms. The first form is applications that provide video analytics of scenes in real time. This form of the application class is driven by the emergence of augmented reality (AR) displays such as HoloLens (www.microsoft.com/microsoft-hololens) and Google Glass (https://developers.google.com/glass), where the mobile device must continuously process video feeds in real time to identify interesting objects in the scene and then perform some action, such as overlaying information on those objects. The second form is applications that migrate desktop gaming to mobile devices, where the mobile device must generate numerous high-resolution video frames in real time.

In both of these cases, the mobile device must either process video frames (for the AR use case) or generate numerous video frames (for the gaming use case) in real time. Neither use case requires a large dataset (because the knowledge required to process or generate the video feeds is relatively small), but both require a very large amount of computational power that might not be available on a mobile device. Indeed, even on desktop machines, processing and generating the high-resolution video feeds (such as at 4K resolutions) might require high-end CPU and GPU hardware (or even multiple machines working together in parallel). Even if

the AUTHORS



Rajesh Krishna Balan is a professor of information systems at the Singapore Management University, where he is also a director of the LiveLabs Urban Lifestyle Innovation Platform. His research interests include mobile systems, power management, and usability. Balan received his PhD in computer science from Carnegie Mellon University. Contact him at rajesh@smu.edu.sg.

Jason Flinn is a professor of computer science and engineering at the University of Michigan, Ann Arbor, where he is also the director of the Software Systems Laboratory. His research interests include operating systems, distributed systems, and mobile computing. Flinn received his PhD in computer science from Carnegie Mellon University. Flinn is a fellow of the ACM, and his research has been recognized with an NSF Career award. Contact him at jflinn@umich.edu.

the computation is performed locally on the mobile device, the energy cost to run the mobile CPU and GPU at continuous full capacity will quickly exhaust the mobile device's battery capacity and generate a large amount of heat.

However, the lack of local computation power and battery capacity, coupled with the need for real-time responses (and the lack of a large dataset), makes these types of applications quite suitable for a cyber-foraging-style solution. Indeed, research prototypes such as Outatime[19] and Kahawai[20] have already shown how cloud rendering on local or remote servers can greatly improve the ability of mobile devices to play desktop-quality games. Other research solutions, such as the one Ha and his colleagues proposed,[21] have also shown how local clouds that epitomize the concept of cloudlets[14] can greatly improve the performance of AR-type applications.

Looking forward, the use of mobile AR and VR displays has the potential to be a compelling use case for cyber foraging. In particular, the heat output of these displays needs to be low, because they are worn on a user's head. Offloading computation to nearby servers just to reduce heat generation might be necessary for practical long-term use. In addition, the running of both AR-driven computer vision applications and high-resolution desktop-quality games on cell phones might require cyber foraging to achieve both the quality and real-time latencies required for such applications.

Fifteen years after the original vision, considerable progress has been made on partitioning, isolation, and other infrastructure issues needed to make cyber foraging a reality. What is needed now is progress along two dimensions: developing the killer apps that this infrastructure enables and lowering cost and other practical barriers that hinder widespread surrogate deployment. P

REFERENCES

1. M. Satyanarayanan, "Pervasive Computing: Vision and Challenges," *IEEE Personal Comm.*, vol. 8, no. 4, 2001, pp. 10–17.
2. J. Flinn, *Cyber Foraging: Bridging Mobile and Cloud Computing*, Morgan and Claypool, 2012.
3. J. Flinn, S.Y. Park, and M. Satyanarayanan, "Balancing Performance, Energy, and Quality in Pervasive Computing," *Proc. 22nd Int'l Conf. Distributed Computing Systems*, 2002; doi: 10.1109/ICDCS.2002.1022259.
4. M.-R. Ra et al., "Odessa: Enabling Interactive Perception Applications on Mobile Devices," *Proc. 9th Int'l Conf. Mobile Systems, Applications and Services*, 2011, pp. 43–56; doi: 10.1145/1999995.2000000.
5. R.K. Balan et al., "Tactics-Based Remote Execution for Mobile Computing," *Proc. 1st Int'l Conf. Mobile Systems, Applications and Services*, 2003, pp. 273–286.
6. E. Cuervo et al., "MAUI: Making Smartphones Last Longer with Code Offload," *Proc. 8th Int'l Conf. Mobile Systems, Applications and Services*, 2010, pp. 49–62.
7. B.-G. Chun et al., "CloneCloud: Elastic Execution between Mobile Device and Cloud," *Proc. 6th ACM European Conf. Computer Systems*, 2011, pp. 301–314.
8. M.S. Gordon et al., "COMET: Code Offload by Migrating Execution Transparently," *Proc. 10th Symp. Operating Systems Design and Implementation*, 2012, pp. 93–106.
9. Y.-Y. Su and J. Flinn, "Slingshot: Deploying Stateful Services in Wireless Hotspots," *Proc. 3rd Int'l Conf. Mobile Systems, Applications and Services*, 2005, pp. 79–92.
10. M. Gordon et al., "Accelerating Mobile Applications through Flip-Flop Replication," *Proc. 13th Int'l Conf. Mobile Systems, Applications and Services*, 2015, pp. 137–150.
11. B.D. Higgins et al., "The Future Is Cloudy: Reflecting Prediction Error in Mobile Applications," *Proc. 6th Int'l Conf. Mobile Computing, Applications, and Services (MobiCASE)*, 2014; doi: 10.4108/icst.mobicase.2014.257722.
12. A. Rudenko et al., "Saving Portable Computer Battery Power through Remote Process Execution," *Mobile Computing and Comm. Rev.*, vol. 2, no. 1, 1998, pp. 19–26.
13. S. Goyal and J. Carter, "A Lightweight Secure Cyber Foraging Infrastructure for Resource-Constrained Devices," *Proc. 6th Int'l Workshop Mobile Computing Systems and Applications* (HotMobile), 2004; doi: 10.1109/MCSA.2004.2.
14. M. Satyanarayanan et al., "The Case for VM-Based Cloudlets in Mobile Computing," *IEEE Pervasive Computing*, vol. 8, no. 4, 2009, pp. 14–23.
15. K. Ha et al., "Just-in-Time Provisioning for Cyber Foraging," *Proc. 11th Int'l Conf. Mobile Systems, Applications and Services*, 2013, pp. 153–166.
16. R. Krishna Balan et al., "Simplifying Cyber Foraging for Mobile Devices," *Proc. 5th Int'l Conf. Mobile Systems, Applications and Services*, 2007, pp. 272–285.
17. J. Flinn and M. Satyanarayanan, "Energy-Aware Adaptation for Mobile Applications," *Proc. 17th ACM Symp. Operating Systems Principles*, 1999, pp. 48–63.
18. D. Narayanan and M. Satyanarayanan, "Predictive Resource Management for Wearable Computing," *Proc. 1st Int'l Conf. Mobile Systems, Applications, and Services*, 2003.
19. K. Lee et al., "Outatime: Using Speculation to Enable Low-Latency Continuous Interaction for Mobile Cloud Gaming," *Proc. 13th Int'l Conf. Mobile Systems, Applications and Services*, 2015, pp. 151–165.
20. E. Cuervo et al., "Kahawai: High-Quality Mobile Gaming Using GPU Offload," *Proc. 13th Int'l Conf. Mobile Systems, Applications and Services*, 2015; doi: 10.1145/2594368.2601482.
21. K. Ha et al., "Towards Wearable Cognitive Assistance," *Proc. 12th Int'l Conf. Mobile Systems, Applications and Services*, 2014, pp. 68–81.