Cloudlet-Based Cyber-Foraging for Mobile Systems in Resource-Constrained Edge Environments

Grace A. Lewis, Sebastian Echeverría, Soumya Simanta, Ben Bradshaw, James Root
Carnegie Mellon Software Engineering Institute
4500 Fifth Ave.
Pittsburgh, PA USA
+1 (412) 268-5800
{glewis, secheverria, ssimanta, bwbradshaw, jdroot}@sei.cmu.edu

ABSTRACT
First responders and others operating in crisis environments increasingly make use of handheld devices to help with tasks such as face recognition, language translation, decision-making and mission planning. These resource-constrained edge environments are characterized by dynamic context, limited computing resources, high levels of stress, and intermittent network connectivity. Cyber-foraging is the leverage of external resource-rich surrogates to augment the capabilities of resource-limited devices. In cloudlet-based cyber-foraging, resource-intensive computation is offloaded to cloudlets – discoverable, generic servers located in single-hop proximity of mobile devices. This paper presents several strategies for cloudlet-based cyber-foraging and encourages research in this area to consider a tradeoff space beyond energy, performance and fidelity of results.

Categories and Subject Descriptors
H.3.4 [**Systems and Software**]: Distributed Systems. D.2.11 [**Software Architectures**]: Domain-Specific Architectures, Patterns.

General Terms
Design, Experimentation, Performance

Keywords
Cyber-foraging, cloudlet, mobile cloud computing, code offload, computation offload, software architecture, cloud computing

1. INTRODUCTION
Mobile applications are increasingly used by first responders and others operating in crisis and hostile environments in support of their missions. These environments are not only at the edge of the network infrastructure, but are also resource-constrained due to *dynamic context*, *limited computing resources*, *intermittent network connectivity*, and *high levels of stress*. Applications that are useful to field personnel include speech and image recognition, natural language processing, and situational awareness. These are all computation-intensive tasks that take a heavy toll on the device's battery power and computing resources.

Cyber-foraging is the leverage of external resource-rich surrogates to augment the capabilities of resource-limited mobile devices [1]. Most existing cyber-foraging solutions rely on conventional Internet for connectivity to the cloud or strategies that tightly couple mobile clients with servers at deployment time. These solutions are not appropriate for resource-constrained environments because of their dependence on multi-hop networks to the cloud and static deployment. Cloudlet-based cyber-foraging relies on discoverable, generic, stateless servers located in single-hop proximity of mobile devices. These characteristics make cloudlets a good match for the characteristics of resource-constrained environments. However, in our research in exploring cloudlet-based and other forms of cyber-foraging we have found that most solutions do not address the challenges of "being at the edge."

The goal of this paper is to present alternatives for cloudlet-based cyber-foraging and set the stage for the need for expanding this work to support the quality attributes required in resource-constrained edge environments. Section 2 presents a short summary of related work in this area. Section 3 describes cloudlet-based cyber-foraging. Section 4 presents five strategies for cloudlet provisioning along with experimental data that shows the pros and cons of each strategy. Finally, Section 5 presents our ideas for new research directions for cyber-foraging to support resource-constrained edge environments.

2. RELATED WORK
Multiple cyber-foraging systems have been developed that differ in terms of the strategy that they use to leverage remote resources — where to offload, when to offload, and what to offload. Where to offload varies between remote clouds and local servers located in proximity of mobile devices. When to offload varies between a runtime decision or an "always offload" strategy. To support runtime offload decisions, one strategy is to manually or automatically partition code into portions that either run on the mobile device or on a remote machine. At runtime an optimization engine — typically targeted at optimizing energy efficiency, performance, or network usage — decides whether the code should execute locally or be offloaded to a remote machine (surrogate). An example of such cyber-foraging system is MAUI [2]. CloneCloud [3] follows the same code partitioning principle but automatically partitions code at the thread level without the need for manual code annotation. Other cyber-foraging solutions assume that the computation-intensive code exists in a remote machine and the cyber-foraging task therefore becomes one of service discovery and composition. HPC-as-a-service [4] is an

ICSE '14, May 31 - June 07 2014, Hyderabad, India

example of this “always offload” strategy. What to offload is what has the most variation, ranging from threads [3] to methods [2] to services [4] to full programs [1], with many other options in between. Our work is based on cloudlets, as described in [1]. Despite all the work in cyber-foraging, our research has showed that (1) there is emphasis on the algorithms to support code offload and state synchronization with minimal focus on software architecture and quality attributes beyond energy efficiency and performance, (2) there is little guidance on how to support quality attributes such as survivability, resilience, trust and ease of deployment, critical in edge environments

3. CLOUDLET-BASED CYBER-FORAGING

Cloudlets are discoverable, generic, stateless servers located in single-hop proximity of mobile devices, that can operate in disconnected mode and are virtual-machine (VM) based to promote flexibility, mobility, scalability, and elasticity [1]. In our implementation of cloudlets, applications are statically partitioned into a very thin client that runs on the mobile device and a computation-intensive server that runs on the cloudlet. A reference architecture for cyber-foraging is presented in Figure 1. The main elements of the architecture are the *Mobile Client* and the *Cloudlet Host*. A *Discovery Service* running inside the cloudlet host publishes *Cloudlet Metadata* that is used by the *Cloudlet Client* to determine the appropriate cloudlet for offload and to connect to the cloudlet. Metadata can range from a simple IP address and port to connect to the cloudlet server to complex data structures describing cloudlet capabilities. Every application is composed of a *Cloudlet-Ready Client App* that corresponds to the client portion, the *Server Offload Code* that corresponds to the server portion, and the *Client App Metadata* that contains information that is used by the cloudlet client and the cloudlet server to negotiate and carry out the code offload process. Once a cloudlet is identified for offload, the cloudlet client sends the server offload code and client app metadata to the *Cloudlet Server*. The cloudlet server then deploys the server code inside a *Guest VM* inside the *VM Manager*. The server offload code can range from provisioning instructions, to source code, to application packages, to complete VMs. Once the deployment is complete, the cloudlet server is notified that the server is ready for execution and the client app is launched.

4. CLOUDLET PROVISIONING

In addition to cloudlet discovery, a key aspect of cloudlet-based cyber-foraging is cloudlet provisioning—transferring and starting the server code on the cloudlet so that it is ready to use by the client running on the mobile device. In the original cloudlet proposal, cloudlet provisioning is done via VM synthesis. In this approach, an application overlay that corresponds to the server portion of a client-server application is created by calculating the binary difference between a base VM image file and the VM image file after installation of the sever on the base VM image. The overlay is carried on the mobile device and transferred at runtime to a discovered cloudlet, where it is applied to the base VM image so that the resulting VM image corresponds to the running server. The full implementation is described and analyzed in [5].

4.1 Previous Work: VM Synthesis and Application Virtualization

One of the main problems with the original proposal for VM synthesis is the large size of the overlays that have to be transferred from the mobile device to the cloudlet at runtime. As reported in [5], the size of the overlay in our experiments ranged from 43.55 MB for a Windows-based face recognition application to 176.23 MB for a Linux-based speech recognition application. The sizes of these two application overlays go up to 172 MB and 343 MB respectively if the overlay includes the memory snapshot in additional to the disk overlay for quicker startup time (Prototype 2 in [5]). Since then, we have added two optimizations to the VM synthesis prototype in an attempt to reduce overlay size as well as application-ready time. The first optimization is pipelining so that overlay decompression is done incrementally as opposed to having to wait until the complete overlay is received. The overlay is compressed using LZMA2 with the XZ stream compression format. At runtime, the compressed overlay is sent in chunks to the cloudlet. Each chunk is placed in a queue, decompressed, and appended to a file. The second optimization is the use of QEMU copy on write 2 (qcow2) as the VM image file format. The advantage of qcow2 is that there is no need to use xdelta to calculate the binary difference between the complete VM and the base VM because the qcow2 file already corresponds to the changes with respect to the base VM. This means that there is no need for extra processing after decompression. However, as can be seen in Table 1, even though application-ready time (time between start of cloudlet provisioning and acknowledgement of server start) improves by ~50% on average compared to the times reported for Prototype 2 in [5], the overlay size is still large. This is a problem given that our experiments confirm that network payload size is directly proportional to energy consumption as has been stated by many others.

We started exploring application virtualization as a way to decrease payload size, which uses an approach similar to OS virtualization, by “tricking" the software into interacting with a virtual rather than the actual environment. To accomplish this, a runtime component intercepts all system calls from an application and redirects these to resources inside the virtualized application. In this approach, what is sent to the cloudlet at provisioning time is an application package that gets deployed into a VM that matches the OS of the virtualized application. The full implementation is described, analyzed, and compared to VM synthesis in [6]. Although payload size is on average 25% of the size of an overlay in VM synthesis, it is still challenging for use in resource-constrained environments.

4.2 Alternate Provisioning Strategies

After conducting a systematic literature review in the area of architecture strategies for cyber-foraging, as well as our prototyping experience, we have noted that even though there are many clever and sophisticated algorithms for code offload, there is very little emphasis on quality attributes beyond energy efficiency, performance and network usage. Although these quality attributes are important in resource-constrained edge environments, there is little discussion of cyber-foraging when there is intermittent or no network connectivity to the cloud. What is required in resource-constrained environments is rapid provisioning and deployment of cloudlets, discovery of available capabilities, and support for disconnected operations. While we believe that cloudlets are best suited for these environments, we are looking at alternate strategies for cloudlet-based cyber-foraging that instead of trying to get the computation from the mobile device to the cloudlet at runtime, they focus on moving industrial cloud computing and mobile computing practices to the edge.

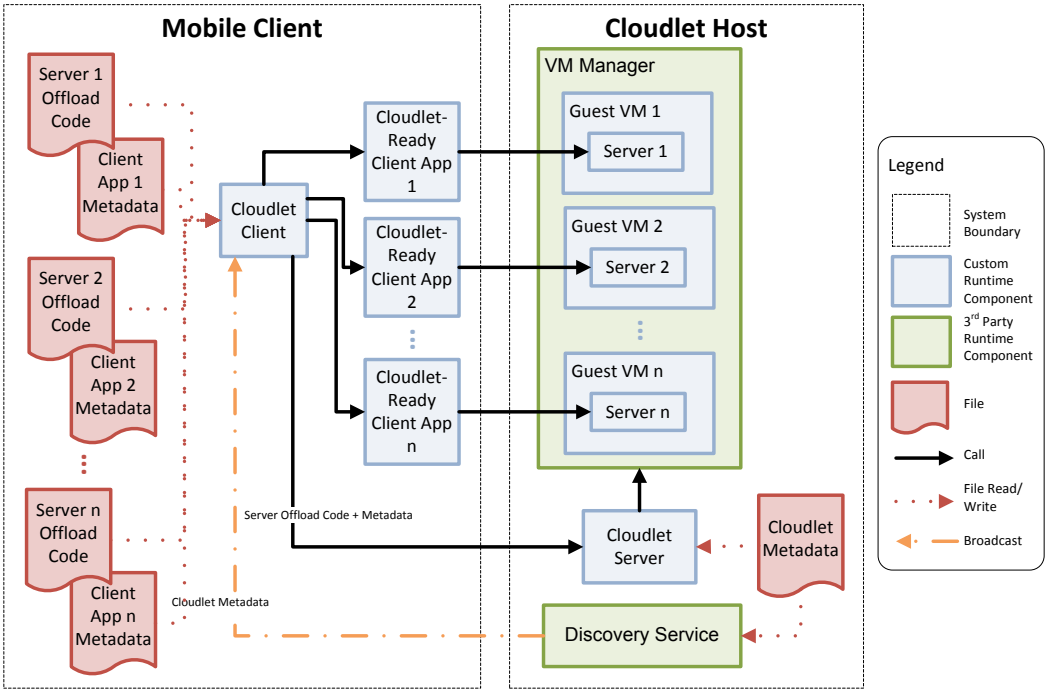

Figure 1. Reference Architecture for Cloudlet-Based Cyber-Foraging

4.2.1 Cached VM

In Cached VM the cloudlet is pre-provisioned with VM images that correspond to capabilities that match the client apps on the mobile device. Each VM image file has a unique service identifier. At runtime, the mobile device instructs the cloudlet to start the VM that corresponds to the service for the launched client app.

4.2.2 Cloudlet Push

In Cloudlet Push, the cloudlet is not only pre-provisioned with VM images for mission-specific capabilities, but also the corresponding mobile client apps. At runtime, the mobile device queries the cloudlet for available capabilities, similar to accessing an app store. The cloudlet pushes the selected client app to the mobile device and then starts the corresponding VM.

4.2.3 On-Demand VM Provisioning

In On-Demand VM Provisioning a commercial cloud provisioning tool is used to "assemble" a VM. Our implementation uses Puppet from www.puppetlabs.com. At runtime, the mobile device sends a provisioning script to the cloudlet. The cloudlet executes the provisioning script to construct and start an appropriate VM.

4.3 Quantitative and Qualitative Comparison of Cloudlet Provisioning Strategies

To perform a quantitative and qualitative comparison of the five different cloudlet provisioning strategies, we conducted a set of experiments using three computation-intensive applications: face recognition (FACE), speech recognition (SPEECH), and object recognition (OBJECT). We used a Galaxy Nexus with Android 4.3 as a mobile device and a Core i7-3960x based server with 32 GB of RAM running Ubuntu 12.04 as the cloudlet. We created a self-contained wireless network (using Wi-Fi 802.11n at 2.4 GHz, 65 Mbps) to be able to isolate network traffic effects. Energy was measured using a PowerMeter from Monsoon Solutions. The results of these experiments are shown in Table 1. The table shows that in general the alternate provisioning strategies consume less energy because payload size is smaller, which in turn leads to shorter and more consistent application-ready times across applications. In Cached VM the payload size is very small (service ID) and application-ready time is basically the time that it takes to start the corresponding VM. In Cloudlet Push the payload is small (client app from cloudlet to mobile device) and the application-ready time is the time that it takes to install the app on the mobile device. In On-Demand VM Provisioning the payload is very small (Puppet provisioning script) but the application-ready time is longer (similar to VM Synthesis times) because it corresponds to the time that it takes to assemble the VM according to the script and then start that VM. This also contributes to higher energy consumption because we measure the energy consumed during the complete cloudlet provisioning process. For the Linux applications the energy consumption is still lower because the client is idle instead of sending data. For Windows applications this is not the case because the application-ready times are much longer because the installation processes are more complicated. However, as shown in Table 2, the tradeoff is that these alternate strategies rely on cloudlets that are pre-provisioned with server capabilities that might be needed for a particular mission, or that the cloudlet is connected to the enterprise, even if just at deployment time, to obtain the capabilities. We argue that this requirement is not unreasonable in edge environments and that it makes cloudlet deployment in the field easier and faster while leveraging the state of art and best practices from the cloud computing industry. A pre-provisioned-VM-based solution also promotes resilience and survivability by supporting rapid live VM migration in case of cloudlet mobility, discovery of more powerful or less-loaded cloudlets, or unavailability due to disconnection or disruption. It supports scalability and elasticity by starting and stopping VMs as needed based on number of active users (which is typically bounded in edge environments because group size is known). In addition, the request-response nature of many of the operations needed in the

field also lends itself to an asynchronous form of interaction in which the cloudlet can continue processing and send results back to a mobile device (directly or by re-routing) as network conditions change.

5. NEW RESEARCH DIRECTIONS

Cyber-foraging in resource-constrained environments would greatly benefit from moving cloud computing concepts and technologies closer to the edge so that surrogates, even if disconnected from the enterprise, can provide offload capabilities that can work at the edge. We would like to motivate research that takes an architectural approach to cyber-foraging and addresses a larger tradeoff space that includes disconnected operations, resiliency, survivability, ease of deployment, and trust. The work presented in this paper represents a step in that direction.

6. ACKNOWLEDGMENTS

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. This material has been approved for public release and unlimited distribution (DM-0000774).

REFERENCES

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

[2] Cuervo, E., Balasubramanian, A., Cho, D.-K., Wolman, A., Saroiu, S., Chandra, R., Bahl, P. 2010. MAUI: Making Smartphones Last Longer with Code Offload. In: *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys '10)*, pp. 49–62. ACM, New York.

[3] Chun, B., Ihm, S., Maniatis, P., Naik, M., Patti, A. 2011. CloneCloud: Elastic Execution between Mobile Device and Cloud. *Proceedings of the 6th Conference on Computer Systems (EuroSys '11)*, pp. 301–314. ACM, New York.

[4] Duga, N. 2011. *Optimality Analysis and Middleware Design for Heterogeneous Cloud HPC in Mobile Devices*. Doctoral Thesis. Addis Ababa University.

[5] Simanta, S, Lewis, G., Morris, E., Ha, K., and Satyanarayanan, M. 2012. A Reference Architecture for Mobile Code Offload in Hostile Environments *Proceedings of the Joint Working IEEE/IFIP Conference Software Architecture (WICSA) and European Conference on Software Architecture (ECSA)*, pp.282 - 286.

[6] Messinger, D., Lewis, G. 2013. *Application Virtualization as a Strategy for Cyber Foraging in Resource-Constrained Environments* (Technical Report CMU/SEI-2013-TN-007). Pittsburgh: Software Engineering Institute, Carnegie Mellon University.

Table 1. Experiment Data for Cloudlet Provisioning Strategies

Applications	Optimized VM Synthesis			Application Virtualization			Cached VM			Cloudlet Push			On-Demand VM Provisioning		
	(1)	(2)	(3)	(1)	(2)	(3)	(1)	(2)	(3)	(1) *	(2)	(3)	(1) *	(2)	(3)
FACE (Windows)	55	53.4	57.8	14	14.3	10.5	0.00	8.2	10.3	0	7.9	13.8	0	112.7	129.1
OBJECT (Linux)	332	175.7	333.3	29	21.9	24.5	0.00	11.6	13.5	0	11.7	16.9	0	211.0	244.0
SPEECH (Windows)	194	85.9	175.5	66	62.5	66.6	0.00	12.2	14.7	0	12.8	18.2	0	237.6	269.2
SPEECH (Linux)	147	99.0	172.5	68	38.3	54.2	0.00	12.2	14.9	0	12.8	18.2	0	94.1	109.3

Columns under each strategy are (1) Payload Size (MB), (2) Application-Ready Time (s), and (3) Client Energy (J)
* Size of payload is less than 1KB

Table 2. Qualitative Comparison of Cloudlet Provisioning Strategies

	Optimized VM Synthesis	Application Virtualization	Cached VM	Cloudlet Push	On-Demand VM Provisioning
Cloudlet Content*	Exact Base VM	VM compatible with Server code	Service (VM) repository	Repository of paired VMs (Server code) and Client Apps	• VM provisioning software • Server code components
Mobile Device Content**	• Application Overlays • Client Apps	• Virtualized server code • Client Apps	Client Apps	None	• VM provisioning scripts • Client Apps
Payload	Application Overlay	Virtualized Service Code	Service ID	Client Apps	VM Provisioning Script
Advantages	Cloudlet can run any server code that can be installed on a Base VM	Portability across OS distribution boundaries	Supports server code updates as long as service interface remains the same	Supports most client nodes with distribution at runtime	Server code can be assembled at runtime
Constraints	Requires exact Base VM which limits distributions and patches	All server code dependencies have to be captured at packaging time	Cloudlet is provisioned with service VMs required by client apps (or has access to them)	Cloudlet has a client app version that matches mobile client OS version	Cloudlet has all required server code components (or access to them)

* In addition to Cloudlet Server
** In addition to Cloudlet Client. Client Apps include Metadata.