Conceptual Approach Towards Stateful Computation Offloading in Resource Constraint Android Devices

P Vekneswaran[1#] and NR Dissanayake[1]

[1] *Informatics Institute of Technology, No 56, Ramakrishna Road, Colombo 6, Sri Lanka*
[#] prathieshna.2016281@iit.ac.lk

Abstract— *Cyber Foraging is an approach to resolve the performance and resource limitations of small portable hand-held devices, through offloading heavy processes to nearby stationary devices called Surrogate Devices, which have more computation capabilities. The way the Cyber Foraging is mostly tackled by invoking a remote method in the Surrogate Device – which contains the application related information as a pre-requisite – at the time of local invocation, in a state-less manner. This limits the possibilities where the Cyber Foraging can be used in state-full context, which can provide maintenance of state which will help the developer to transmit state of objects used inside the offload candidate so it won't differ from local execution and remote execution. In order to use in a stateful manner, researchers have used a virtualisation approach, which is a resource consuming approach. As an alternative, we propose a conceptual solution to transmit state from mobile platform to the surrogate platform, without having to deploy a mobile OS virtual machine into the surrogate environment. The concept will address all the drawbacks of virtualisation and provide the similar benefits at a lower resource cost in the Surrogate end.*

Keywords—**Cyber Foraging, Distributed Computing, State-full Offloading**

I. INTRODUCTION

This section gives a brief background to the domain cyber foraging, specifying the problem we are focusing on and the motivation towards proposing a conceptual approach towards stateful computation offloading and the methodology used for the research. Followed by existing work, proposed approaches, limitations, conclusions and future work.

A. Background

1) Cyber Foraging: Since then there has been various attempts to fulfil Mark Weiser's vision (Weiser, 1991); however, these mobile devices are coupled with some limitations like limited processing power and battery standby. Cyber Foraging, which was a term coined by M. Satyanarayanan, is an attempt to offload process from a portable resource constraint platforms of mobile devices to a stationary resource rich platform, called a surrogate. Following his approach there have been various attempts towards a perfect cyber foraging solution, which is both efficient and developer friendly.

2) Offloading Approaches used in Cyber Foraging and their Limitations: There are two computation offloading approaches available for cyber foraging: 1) State-less computation offloading and 2) State-full computation offloading.

State-less offloading is where the surrogate is not aware of the state of the application or objects in the heap memory. Surrogate will execute the offloaded method in a method level where the parameters are passed and method will be executed in a static context. Most of the cyber foraging approaches do not maintain state during the offload, as a consequence granularity of the off loadable component is limited to the method.

There are only very few significant approaches, which support state-full offloading. They use virtualisation, where the mobile operating system is replicated in a remote location and the mobile device and the surrogate share the application state among each other, which is costly on the surrogate end, making it unattractive towards the surrogate owner. This also creates a dependency between surrogate and the mobile device.

In order to avoid these adverse impacts, we propose a state-full offloading approach, which does not create unnecessary dependencies and does not involve Virtualisation.

B. Methodology

The existing approaches and their specifications were identified from a literature survey. Almost 57 preliminary researches has been conducted in the domain of enabling cyber foraging (Lewis & Lago, 2015). The survey published by Lewis & Lago covers majority of the approaches at the time of their writing. It was helpful in revisiting the approaches, and also recent approaches had been surveyed and considered within the scope of this research.

We took the liberty of referring our previous work in the domain, targeting towards a better practical solution into consideration, as empirical research. We had experimented with a state-less architecture, and we are currently focusing on supporting state-full offloading without virtualisation, which was one of the major

limitation addressed in our earlier proposed architecture. (P Vekneswaran, 2016)

II. Existing Work

Even though there are significant amount of researches done in the domain, there is only a handful of approaches that maintains state while offloading; and all of them are virtualisation based solutions where maintaining the state is vital to react upon each invocation.

A. AMCO

The mobile application developer declaratively specifies the suspected energy consumption hotspots in a mobile application. Based on this input, AMCO then automatically transforms the application to enable it to offload parts of its functionality to the cloud. The offloading is exceedingly versatile, being driven by a runtime framework that powerfully decides both the state-to-offload and its exchange instrument in light of the execution condition set up. In addition, the system continuously improves with a feedback-loop mechanism.

To mark hotspot components, AMCO provides a Java annotation @OffloadingCandidate; this information can also be specified through an XML configuration file. Based on this input, an analysis engine first checks whether the specified component can be offloaded as well as any of its sub-components (i.e. successors in the call graph). The engine additionally ascertains the program state, to be exchanged between the remote and local segments that would should be exchanged to offload the execution of both the whole segment or of any of its sub-segments. A bytecode enhancer at that point creates the checkpoints that spare and re-establish the ascertained state for the whole hotspot segments and additionally for each of its subcomponents (Kwon & Tilevich, 2013).

It is easier to develop cyber foraging enabled application on top of AMCO because it is just a matter of annotating the offload candidates or migrating the entire processes to the cloud. But it also adverse impacts such as cost of sharing the entire state, tracking state modifications also offloading everything to the could not be beneficial at all times.

B. ROAM

ROAM is a JAVA application framework that can assist developers to build Java Based multi-platform applications, which run on heterogeneous devices; and allows a user to move/migrate a running application among heterogeneous devices without any significant effort by the user.

In nutshell, the ROAM agent on the source device first negotiates with the ROAM agent on the target device. The negotiation involves exchanges of the target device capabilities needed by each application component, and the code base URL where the ROAMlet component byte code can be downloaded from. Based on the exchanged information, the ROAM agent decides the appropriate adaptation strategy for each component. The ROAM agent on the target device downloads the necessary ROAMlet class byte code from the HTTP server for all application components that will be instantiated on the target device. The ROAMlet on the source device serializes its execution state and sends it to the ROAM agent on the target device. The ROAM agent may perform execution state transformation if an application component is transformed or dynamically instantiated. The ROAM agent instantiates the ROAMlet on the target device. (Hao-hua Chu, 2003)

ROAM doesn't target computational offloading directly but does focus on task migration aspect in depth. What ROAM is trying to achieve is pretty far fetch because it is attempting to tackle pervasive computing first rather than Cyber Foraging itself. Server components must be manually written so according to the device context the code/service will be retrieved from the server. Even though it mentions about the heterogeneous devices, they are targeting Java based operating systems all run on JVM. So, transforming state is straight forward.

C. Clone-Cloud and Cloudlet

Cloud computing alludes to a style of figuring where online assets and applications are accessed by clients through a web browser, yet the real software and data are put on remote servers. As the client's point of view, the client does not have to buy, oversee or maintain the technology "in the cloud". Intel's thought is to take the mobile device that has lower performance and clone the whole arrangement of information and applications onto the cloud, at that point sync the two. It is then likewise conceivable to dole out graphics and processing power to the task in the cloud and give an apparently elite with full PC-like experience to the present smartphone or netbook with no change to the client experience and interface. (Chun, Ihm, Maniatis, Naik, & & Patti, 2011)

Clone-Cloud and Cloudlet approaches don't fall too far away from each other. Cloudlet is a concept to deploy an in-box cloud in house, which is self-managed and does not need any professional attention. A cloudlet is deployed for few users to share the computing resource via the high-speed LAN network, rather than using the distant public cloud on the Internet in order to shorten the latency for real time application.

Even though it's easier to develop just like AMCO, Clone-Cloud requires heavy modifications to the Android Device kernel to support this infrastructure.

D. Collaborative Applications with Mobile Cloud

This contains an architecture and a paradigm for developing collaborative applications with minor modifications to today's mobile and cloud computing infrastructures. This approach is focusing on cloning the kernel requests and responses in the cloud to maintain state. Also has an offload advisor to evaluate the effectiveness of offloading and make the choice to offload or not. (Yu-Shuo Chang, 2012)

This also requires hosting and has the same drawbacks as of Clone Cloud and Cloudlet.

III. Proposed Approach

In order to enable less developer effort in cyber foraging enabled mobile applications development, we think that the assistance provided by the development framework to the developers is an important fact. (P Vekneswaran, 2016) Aspects (JBoss, 2016) reduce the build process immensely, by reducing the need to write separate code for the off loadable components, which will be discussed in the following section. Also, we made improvements for achieving computation offloading, to support state transmission to other VMs.

A. Aspect based Annotation

Candidate methods, the resource intensive components of the code, recommended for offloading can be scattered all over the application, thus the application has to invoke the decision-making engine every time before such method is invoked. Knowledge of those offload method calls is simply irrelevant to the business logic in the class. In such scenario, we propose to use Aspect Orientation in mobile application code, to identify the compute intensive tasks all around the application, and while using annotations won't interfere with the business logic or the object orientated design and architecture of the application. (P Vekneswaran, 2016)

The annotation is the key of our proposed approach, which helps the developer to specify the identified intense tasks in the application to state, which affects the execution. In this approach, there are already predefined advices, which need to be injected during the compilation phase. Here the advice is the code that is injected into the class file; typically, which needs to be inserted before, after, or instead of the target method. When the method is marked using the "offloadMethod" annotation, AspectJ will generate the boilerplates that is necessary for the annotated method.

In our approach, we suggest annotating the potential offloading candidate methods in classes, using the "offloadMethod" along with the name of the objects which states need to be transferred to the Surrogate using Aspect orientation. Just using one annotation, identifying different methods with different behaviours is not possible. Therefore, it is necessary to be able to annotate a single method using multiple annotations as a part of the framework we propose; so, the developer can create his custom defined aspects through creating new annotation interfaces using default templates provided.

B. Development Process and Limitations

When using the annotations, the developer should identify the methods and objects that is referenced inside the context that consists of the compute intensive components that has be considered by the decision-making engine during the runtime, weather it needs to be offloaded or not. This requires the developer to have a basic knowledge of the application source, states of the objects that is required for offloading, and the flow of execution.

Identified intensive methods cannot reside inside Android Activities. These methods should not attempt to read hardware specific sensor data inside their scope, which will cause errors during the runtime. Alternative is to read the sensory data and pass it as a state. The intensive components should be added to a separate java class for it to be executed correctly in the surrogate environment. If the source is already separated this step can be ignored.

These candidate methods should be marked by the developer by adding annotations along with the object details, which the state needs to be preserved. During the build time boilerplate codes will be added by the AspectJ runtime, which is included in the framework. There are predefined aspects, which the developer can use, or if the developer is looking for some distinct characteristics, he can modify the aspects accordingly. Afterwards developer can build the application and produce the APK, which is ready to be installed in the mobile devices.

The surrogate service should be running at the time when the application is about to execute the compute intensive tasks. According to configuration, the mobile application will find the surrogate device and attempt to connect to the service and offload. The surrogate service will then check if the mobile application is already in its repository of packages. If not, it will pull the source from the mobile device. As the components are sent automatically to the surrogate, there is no need for the developer to manually deploy the components explicitly. This is a one-time process, and afterwards any mobile device running the similar application can offload to that particular surrogate device, without pulling the source to the surrogate. Once the source is loaded, the surrogate will execute the task copying the objects and their states to its heap and return the result back to the mobile device with the updated heap state. If it is not beneficial to offload, the mobile device will execute the task normally without offloading to the surrogate.

IV. Evaluation

The proposed solution will have the following advantages over the existing development approaches.

It eliminates the need for writing two separate codes to do the same task in the mobile and surrogate to preserve state, or use virtualisation as an alternative to enable cyber foraging in new applications. Also, no need to modify the existing source code, just need to annotate the offload candidate methods and meta information required to process the information. This will cut down the time required by the developer to enable cyber foraging in mobile applications. The Aspect based approach will allow the developer to plug and play his own logic in the decision-making engine, communication protocol etc. according to his requirements giving him/her full freedom to customize the outcome. When building the application, the AspectJ runtime will handle the generation of the necessary boilerplate codes, which will further cut down the developer cost. In comparison to the existing approaches that is discussed in Section 3, the proposed solution in this paper has majority of the development process automated and easily configurable through the build scripts. This approach will also give developers, enough customisation options as well. Also without hosting the entire mobile operating system the lightweight service will compute the JAVA components.

V. Conclusion and Future Work

Even though cyber foraging can be easily achieved through virtualization, there are other factors such as functional overhead, which have adverse impact on the energy consumption and performance. We can conclude that the offload technique we propose is more effective in a preliminary level, and reducing the time consumption for the development by automating most of the build process with the help of AspectJ and Android Development Environment. At the moment doesn't support non-java based platforms. Also, it will not support rapid state changes at the moment.

We expect to further improve the proposed approach and introduce a framework, allowing more common features like integration to the available IDEs, which will reduce the developer effort furthermore, supporting rapid development. Future of this research will extend to supporting the android code such as activities, fragments rather than being only able to offload the JAVA components of the application.

References

Balan, Krishna, R., Gergle, D., Satyanarayanan, M., & Herbsleb., J. (2007). Simplifying cyber foraging for mobile devices. (pp. 272-285). ACM.

Balan, R., Flinn, J., Satyanarayanan, M., Sinnamohideen, S., & Yang, H.-I. (2002). The Case for Cyber Foraging. *Proceedings of the 10th workshop on ACM SIGOPS European workshop* (pp. 87-92). ACM.

Chun, B. G., Ihm, S., Maniatis, P., Naik, M., & & Patti, A. (2011). Clonecloud: elastic execution between mobile device and cloud. (pp. 301-314). Proceedings of the sixth conference on Computer systems.

Flinn, Jason, Park, S. Y., & Satyanarayanan, M. (2002). Balancing performance, energy, and quality in pervasive computing. (pp. 217-226). IEEE.

Gartner. (2016). *Gartner Says Worldwide Smartphone Sales Grew 9.7 Percent in Fourth Quarter of 2015*. Retrieved April 16, 2016, from http://www.gartner.com/newsroom/id/3215217

Hao-hua Chu, H. S. (2003). Roam, a seamless application framework. *The Journal of Systems and Software* .

JBoss. (2016). *Chapter 1. What Is Aspect-Oriented Programming?* Retrieved March 30, 2016, from http://docs.jboss.org/aop/1.0/aspect-framework/userguide/en/html/what.html

Kwon, Y.-W., & Tilevich, E. (2013). Reducing the Energy Consumption of Mobile Applications Behind the Scenes. IEEE.

Lewis, G. A., & Lago, P. (2015). A Catalog of Architectural Tactics for Cyber-Foraging. ACM.

P Vekneswaran, D. N. (2016). Annotation based Offload Automation Approach for Cyber Foraging Frameworks. *9th International Reserch Conference.* KDU.

Satyanarayanan, M. (2001). Pervasive computing: Vision and challenges. *Personal Communications, IEEE 8.4* (pp. 10-17). IEEE.

Weiser, M. (1991). The computer for the 21st century. (pp. 94-104). Scientific american 265.3.

Yu-Shuo Chang, S.-H. H. (2012). Developing Collaborative Applications with Mobile Cloud. *Journal of Internet Services and Information Security*.