International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 6, Issue 8, August 2016)

Mobile Code Offloading: A Review

Praseetha V. M.[1], S. Vadivel[2]
[1]St.Joseph's College of Engineering & Technology, Kerala, India
[2] BITS Pilani Dubai Campus, Dubai International Academic City, Dubai

***Abstract*— Mobile devices, especially smart phones, are inevitable to human life because of its communication capabilities irrespective of place and time. Mobile Cloud Computing is a rapidly growing research area which is a combination of mobile computing and cloud computing. As resource scarcity is the major issue with mobile devices, cyber foraging has been considered as a technique by which the resource poor mobile devices can offload their computation intensive tasks to some other stronger machines in the cloud. The network bandwidth and latency plays an important role in this case. This paper is a review on the existing code offloading techniques in mobile cloud computing.**

***Keywords*— Mobile Cloud Computing, Code Offloading, Cloud Computing, Virtual Machine, Cyber Foraging, latency**

I. Introduction

Mobile technology is growing very fast and todays mobile devices are very good in memory, processing power, connectivity, energy consumption etc. But still they are found to be inefficient in executing media rich and computer vision tasks. Many mobile applications like augmented reality, healthcare sensing and analysis are not reaching their full potential since they are too compute intensive and too energy intensive. Applications like speech recognition [1] [2] [3], optical character recognition, language translators, image processors [4] [5], video processing [6], online games and wearable devices require powerful computational facilities. These applications require considerable energy consumption and good computing power which restricts the developers to implement them for mobile devices Mobile cloud computing is considered as the solution for resource limitations of the mobile device in which the compute intensive tasks are offloaded to the cloud. These tasks are processed in the cloud with the help of resource rich devices in the cloud and the result is given back to the mobile device. Thus the limitations of the mobile device are alleviated by using various augmentation strategies. The different augmentation strategies include screen augmentation, energy augmentation, storage augmentation and application processing augmentation of mobile devices [7]. Thus with MCC mobile devices can experience a variety of cloud services at low cost [8] [9].

Cyber foraging [10] [11] [3] [12] has been employed to leverage external resources to augment the capabilities of resource limited mobile devices. Cyber foraging is a term introduced by Satyanarayanan [10] to refer to the process of outsourcing computational tasks to remote servers in the close proximity. Multiple cyber-foraging systems have been developed that use different strategies to leverage remote resources - where to offload, when to offload, and what to offload [13].

Since the clouds are situated at a distant location and in such cases offloading to the cloud is not seemed to be an ideal solution since latency and jitter affect the performance. When executing the computationally intensive mobile applications, the best performance can be achieved by dynamically partitioning the applications in a better way between mobile device and cloud. Researchers were trying to develop new offloading schemes which are efficient both with time and energy. As a result many architectures were proposed by researchers and in this paper we are trying to explain about those proposed architectures.

II. Different Architectures

A lot of architectures have been developed for application offloading. Different offloading frameworks outsource the computational task of the mobile device at different granularity levels. There are mainly two different application portioning approaches [16]. They are static partitioning and dynamic partitioning. The compute intensive components of the mobile application are separated only once with the static application partitioning approach. The application partitioning can be done casually or periodically in dynamic partitioning.

The compute intensive components are offloaded with the help of runtime profiler and solver, which are activated at necessary situations, in casual partitioning. In periodic partitioning, a run time optimization mechanism is used to periodically evaluate the utilization of computing resources on mobile device.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 6, Issue 8, August 2016)

The profiler evaluates the availability and requirement of computing resources for the mobile application and in case of unavailability of sufficient resources, the application will be partitioned and the computational intensive components are offloaded to a remote server node at runtime for processing. After successful completion the result is returned to the application running on the mobile device.

A. Mobile Cloud Hybrid Architecture (Mocha)

This architecture has been developed at University of Rochester as a solution to support massively-parallelizable mobile cloud applications [17]. MOCHA is an offloading framework developed by T. Soyata et. al for applications which demand real time response such as face recognition applications and object recognition in battlefield [18]. The highly accurate face recognition algorithms which do all the required image processing steps, typically require intensive computations. To overcome the latency problem with the cloud, the authors use special-purpose inexpensive compute-box with the capability of massively parallel processing (MPP). A mobile-cloudlet-cloud framework is used in this architecture and algorithms were developed to minimize the overall response time. The mobile devices such as laptops, smartphones etc. are connected to the cloudlet (which supports multiple network connections such as 3G/4G, Bluetooth, and WiFi), which in turn connect to the cloud. The mobile device is responsible for capturing the image and sending the captured image to the cloudlet for preprocessing. If the cloud is situated very near to the mobile device and if the network latency is very small, the mobile device can send the captured image directly to the cloud. But sending the raw image to the cloud is very network intensive and because of this reason some preprocessing steps such as feature extraction should be done on the cloudlet or at the mobile device itself.. after preprocessing a subset of data is send to the cloud for processing and finally the result is given back to the mobile device.

In this architecture mobile devices are transmitting data to the cloudlet where it is stored and the cloudlet updates a profile of network latencies and their variation to reach different cloud servers. Thus to minimize the overall communication latency a dynamic task partitioning is done by the cloudlet to select the best server(s). This helps to optimize the quality of service (QoS). Authors found that a speed-up of approximately 8% is possible when using a cloudlet compared with no cloudlet.

B. VM Based Cloudlets

In [19] Satyanarayan et al. proposed a solution known as cloudlet which is a new architectural element for cyber foraging that represents the middle tier of a 3-tier hierarchy: mobile device –cloudlet – cloud [20]. They define cloudlet as a decentralized and self-managed resource rich computer with few users at a time and they state that ‘‘internally, a cloudlet resembles a cluster of multi-core computers, with gigabit internal connectivity and a high-bandwidth wireless LAN’’. They are widely dispersed on designated areas or places, located closer to the mobile device and connected to a larger cloud server via the Internet. Instead of 3G/LTE the mobile devices use Wi-Fi to connect to the nearby cloudlet because of its lower power consumption, good connection speed and lower latency [21].

As explained in [22] the requirements for both the cloud and cloudlet include:

- strong isolation between untrusted user-level computations
- authentication, access control and metering mechanisms
- dynamic allocation of resources
- ability to support a wide range of user applications

Through using the low latency, high Bandwidth and one-hop wireless access to the cloudlets mobile device gets a real-time interactive response. During runtime, the application discovers an available cloudlet and then offloads the computation-intensive code to it. At this point the only communication is between the mobile device and the cloudlet. When the mobile device moves away from the cloudlet, it could switch over to a degraded service mode that connects to a distant cloud server-or at worst case-even operate offline. Since the cloudlets are very similar to small data centers and are widely dispersed on designated areas or places such as a coffee shop or hospital, they are self-managed and are with decentralized ownership. Also, a cloudlet would only contain cached data that is available elsewhere, so that the loss of a cloudlet would not be disastrous.

Cloudlets are a way to implement cyber foraging. There are many cloudlet architectures proposed. Here, we have studied VM Based Cloudlets [19]. These cloudlets are an effective way of serving multiple users simultaneously in an isolated environment. Each user is allocated a separate VM specific to his application on the cloudlet. They follow the principle of “pre use customization and post use clean-up”.

VM Migration and Dynamic VM Synthesis are two major architectures for VM Based Cloudlets. VM Migration architecture pauses a running VM on mobile device, transfers it to the cloudlet and resumes it there. In other words, the memory image of the source server VM is transferred to the destination server without stopping execution [23]. An illusion of live and seamless migration is obtained by copying the memory pages of the VM priorly without affecting the operating system and other applications. Since the code is pre-copied, no code exchange is done during offloading. As far as the mobile device is concerned, VM Migration is heavy and it is time consuming.

Since the performance of dynamic VM Synthesis depends solely on local resources and WAN failure wouldn't affect synthesis, the authors propose to use dynamic VM synthesis. Dynamic VM Synthesis architecture [19] [24]provisions a custom VM on the cloudlet dynamically and the mobile device discovers a cloudlet in its LAN network using any of the discovery protocols. Dynamic VM Synthesis architecture uses virtual machine technology to rapidly instantiate customized service software on a nearby cloudlet [19]. The mobile device acts as a thin client and can access this service over a Wireless LAN. When the mobile device establishes a TCP connection with the cloudlet and authenticates itself, then, it transfers a VM overlay to the cloudlet. The cloudlet applies this overlay to the base VM to generate the Launch VM running the backend server for the user application.

Tactical Cloudlets can be hosted on vehicles or other platforms [25] in proximity of mobile users to

- provide infrastructure to offload expensive computation
- provide forward data staging for a mission
- perform data filtering to remove unnecessary data from streams intended for dismounted users
- serve as collection points for data heading for enterprise repositories.

The first and foremost advantage is the latency reduction compared with the cellular networks as the computation and information reside on a nearby device. The required application can be accessed directly and instantly by the interaction with the cloudlet. Offloading the task to the cloudlet saves money since the expensive data charging in roaming situation is avoided and it also saves mobile device's energy [26]. The most important advantage of cloudlet is its ability to operate even if it is disconnected from the internet.

Since cloudlets are considered as the intermediate offload elements, the mobile device does not need to communicate with the enterprise cloud during the offload operation [27]. The mobile device need to communicate with the offload element which is very closest and it can continue with offloading even when its offload element is completely disconnected from the cloud.

C. Mobile Assistance Using Infrastructure (MAUI)

The motivation behind the development of MAUI is the thought that the major limitation for the future generation mobile devices will be battery life. We know that 3G connections are suffered with long latency and slow data transfer compared to WiFi. Because of this poor performance, the energy consumption will be high in 3G. when the mobile code is offloaded to the cloud, the typical round-trip time(RTTs) to cloud servers are in the order of tens of milliseconds. Instead of this, if a nearby server co-located with a WiFi access point is used, the RTTs will be less than 10ms. Fig.6 shows the difference in energy consumption when WiFi and 3G connections are used.

As remote execution is a solution for the energy needs of mobile devices, lot of attempts have been made in this direction. Most of these attempts can be categorized into two. The first category is fine-grained leading to large energy savings. In this category the programmer should specify how to partition a program and which sub parts should be remotely executed. The second category uses complete process [28] or full VM migration. In this case, since the entire code and program state is sent to the remote infrastructure, the burden on programmers is reduced.

MAUI is a system that achieves the benefits of the two categories of remote execution mentioned above. It enables fine-grained energy-aware code offload and remote execution using the .Net framework [14] which saves energy while minimizing the modifications required applications. MAUI achieves these benefits by using code portability, serialization, reflection and type safety of the managed code environments. Code portability is used to create two versions of a smartphone application-one runs locally in the smartphone and the other runs remotely-by ignoring the difference in the instruction set architecture between the local device and the remote server. The programming reflection combined with type safety is used to identify the methods which can be benefited by remote execution and then extract the program state needed only by those methods. Serialization is used to determine the size and cost of each method profiled by MAUI.

The program portioning scheme of MAUI is dynamic because of continuous profiling of methods and runtime serialization.

MAUI provides a programming environment, Microsoft .NET Common Language Runtime (CLR), where developers annotate which methods of an application can be offloaded for remote execution. Each time a method is invoked and a remote server is available, MAUI uses its optimization framework to decide whether the method should be offloaded. Once an offloaded method terminates, MAUI gathers profiling information that is used to better predict whether future invocations should be offloaded. If a disconnect occurs, MAUI resumes running the method on the local smartphone. Two proxies are generated on the mobile device and the server that handle control and data transfer to decide on which methods to run remotely and which to run locally.

A high-level architecture of MAUI is given in figure. On the smartphone, the MAUI runtime consists of three components: the solver which is an interface to the decision engine, a proxy which handles control and data transfer for offloaded methods, and a profiler which is responsible for collecting the program's energy measurements and data transfer requirements. Four components are there on the server side. The profiler and the server-side proxy which perform similar roles to their client-side counterparts. The decision engine periodically solves the linear program and the MAUI coordinator is responsible for authentication and resource allocation.

MAUI uses the benefits of managed code to reduce the burden on programmers to deal with program partitioning while maximizing the energy benefits of offloading code. MAUI decides at runtime which methods should be remotely executed, driven by an optimization engine that achieves the best energy savings possible.

The developers have experimentally proved that (1) MAUI improves the energy consumption of resource-intensive applications, such as face recognition, by almost one order of magnitude (i.e., a factor of eight) and reduces the execution time by more than a factor of 6; (2) MAUI allows latency-sensitive applications, such as games, to more than double their screen refresh rates; and (3) MAUI enables the development of applications that bypass the limitations of today's smartphone environments.

D. Clone Cloud

The primary deign goal of CloneCloud is to allow fine grained flexibility in program partitioning and making decisions on what to run where. It also aims to provide seamless automatic application partitioning.

CloneCloud [15] is a system that automatically transforms mobile applications to benefit from the cloud. The CloneCloud migrator operates at thread granularity for application migration which is different from the traditional suspend-migrate-resume mechanisms and it allows to execute the native system operations both at the mobile device and at its clones in the cloud. The system is a flexible application partitioner and execution runtime that enables unmodified mobile applications running in an application-level virtual machine to seamlessly offload part of their execution from mobile devices onto device clones operating in a computational cloud. With static and dynamic code analysis the partitioner automatically identifies cost of the code and the partitioning problems are solved using an optimizer. CloneCloud uses a combination of static analysis and dynamic profiling to partition applications automatically at a fine granularity while optimizing execution time and energy use for a target computation and communication environment.

At runtime, the application partitioning is effected by migrating a thread from the mobile device at a chosen point to the clone in the cloud, executing there for the remainder of the partition, and re-integrating the migrated thread back to the mobile device. It in a sense is a clone of your existing smart phone but it will be entirely in the cloud. In otherwords your smartphone will make a copy of itself on remote servers which will be provided by your cellular service provider, and all the processing power your phone has trouble doing on its own will be sent to the servers via the high speed of smart phones web connectivity.

The primary focus of CloneCloud is a client-server service in which data that requires heavy processing is transferred onto external servers to process. The processed data is then retrieved by the mobile device. This would free the mobile device's resources and deplete less of its battery life. The connection is achieved between the OS of the mobile device and of the Cloud. The mobile device contains a Controller and a Replicator component. While the servers contain an Augmenter and a Replicator component. The Replicator, keeps the changes in the mobile device and the cloud synchronized. The Controller of the mobile device starts the change process and updates the results of the device. It also synchronizes states with the Replicator and coordinates the changes. The Augmenter of the clone allows the data to be manageable from an external location. This component returns the results to the device.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 6, Issue 8, August 2016)

Processing speed, battery life and anti-virus protection are the major advantages of CloneCloud architecture.

E. ThinkaAir

ThinkAir [29] is a framework which provide on demand resource allocation and exploit parallelism through which it gives method level offloading. As stated by the authors the key design goals include dynamic adaptation to changing environment, ease of use for developers, performance improvement through cloud computing and dynamic scaling of computational power.

In this framework any method to be offloaded is annotated with @Remote. The method is invoked via the execution controller which is responsible for making offloading decisions. The decision making parameters include the data collected about the current environment as well as that learnt from past executions. ThinkAir defines four other policies which the user can set according to their needs. They are execution time, energy, execution time and energy, execution time, energy and cost. The offloading decision is taken by considering these policies. The execution controller also deals with profiling and communication with the server. The client handler is responsible for code execution. It manages connection, executes code and return the results back.

F. Code Offload By Migrating Execution Transparently (COMET)

COMET [30], built on top of the Dalvik Virtual Machine, is a runtime system to allow unmodified multi-threaded applications to use multiple machines. The system allows threads to migrate freely between machines depending on the workload. COMET leverages the underlying memory model of the runtime to implement distributed shared memory (DSM) with as few interactions between machines as possible. Making use of a new VM-synchronization primitive, COMET imposes little restriction on when migration can occur. Additionally, enough information is maintained so one machine may resume computation after a network failure.

There are two important limitations of this approach. First, COMET may decide to send over data that is not needed for computation. This is often wasteful of band-width and can make offloading opportunities more sparse. The second limitation lies in the kinds of computation demanded by smartphones.

G. Other Works

In [31] P. Angin and B. Bhargava proposed a dynamic performance optimization framework for mobile cloud computing using mobile agent based application partitioning which imposes minimal structural requirements on the cloud. In this framework, before the installation of the application on the mobile device, the application is partitioned statically into two sets of components. These components include a set of agent based application partitions that are offloadable to the cloud for execution and a set of native execution components that are always executed on the device. Like in MAUI, programmer help is needed for correctly identifying the partitions to be offloaded to reduce the processing overhead. The authors performed experiments with two real-world applications and they state that the proposed framework is promising for improved performance and wide adoption in mobile-cloud computing.

In [32] R. K. Balan et al. proposed a framework known as Chroma, which is a tactics based remote execution system. Tactics is a small sized compact declarative form which contain application specific knowledge necessary for making decisions on effective partitioning and placement. A key feature of Chroma is that applications are isolated from the underlying operating system, device characteristics, and resource availability. Authors have proved that the performance of Chroma is comparable to a runtime system that makes perfect partitioning decisions.

D. Kovachev and R. Klamma in [33] proposed a middleware known as Mobile Augmentation Cloud Services (MACS) through which the mobile application can be offloaded and executed on cloud. The main design goal of this framework was to enable transparent computation offloading of elastic mobile applications. It is a services-based middleware which require the computation-intensive parts to be developed as Android services. The code partitioning for local and remote execution is considered as an optimization problem and the middleware offloads the code to the cloud based on the solution of this problem. The authors guarantee 95% energy savings and significant performance gains for computation-intensive applications compared to local execution.

S. Park et al. in [34] proposed an offloading architecture named SOME (Selective Offloading for Mobile Computing Environment) which reduces the computational cost for mobile devices.

This framework splits the JavaScript-based application codes into local or light weight code for client and remote or heavy weight code for server. Here the programmer annotates the code to be offloaded. SOME uses a built-in proxy called Contents Adaptor(CA) which comes in between the mobile client and the outbound servers. When it detects JavaScript functions in the code, the generator is notified, which creates a skeleton process for the mobile client and also the CA sends a lightweight modified webpage to the client. After receiving the lightweight webpage, the client will communicate with the offloading server sharing the variable between the server and the client. Thus instead of executing a heavy function, the client only sends a parameter to the offloading server and receives the results of the function and so the performance of the mobile client is improved.

Odessa, a novel, lightweight, runtime that automatically and adaptively makes decisions on offloading and parallelism for mobile interactive perception applications was proposed in [35]. As described by the authors, when the mobile is disconnected from the server, the Odessa runtime which runs on the mobile device enables Odessa to transparently improve application performance across different mobile platforms. Odessa uses a greedy algorithm and its decision engine uses simple predictors to estimate whether offloading or increasing the level of parallelism of the bottleneck stage would improve performance. The authors have proved that this approach works very well to improve make span and throughput, and incurs negligible overhead.

Cuckoo [36] is a programming model for code offloading with consideration given to intermittent network connection which is designed for Android. It makes the development of smartphone applications that benefit from computation offloading simpler. It also provides a dynamic runtime system which can decide at runtime whether a part of an application will be executed locally or remotely. Cuckoo has been implemented on top of the Ibis [37] High Performance Programming System, which offers an interface for distributed applications. The main objective of Cuckoo is to reduce the energy consumption on smartphones and increase the speed of compute intensive operations.

Cuckoo integrates with the popular open source Android framework and the Eclipse development tool.

Scavenger [38] is another framework which employs cyber foraging. It uses a mobile code approach to partition and distribute jobs. It is using WiFi for connectivity, and also introduces a scheduler for cost assessment which is based on the speed of the surrogate server. With this framework, it is possible for a mobile device to offload to one or more surrogates and it is shown that running the application on multiple surrogates in parallel is more efficient in terms of performance. It does not discuss fault tolerance mechanisms and since its method is strictly about offloading on surrogates and not sharing, it is not really dynamic.

IC-Cloud [39] proposed by C. Shi et.al is a computational offloading framework where the internet connectivity is intermittent and of varying quality. As explained by the authors the three key ideas of IC-Cloud are lightweight connectivity prediction, lightweight execution prediction and usage of these predictions in a risk controlled manner to make offloading decisions. The signal strength and user historical information are used for connectivity-prediction and the execution-prediction mechanism uses machine learning on dynamic program features to automatically, efficiently, and accurately predict the execution time of offloadable tasks, both on the phone and in the cloud. After accurately predicting the future connectivity and execution time, for making the offloading decision, a global optimal solution [40] can be used.

III. Evaluation

Improved service and resource availability, enhancement of security because of dynamic partitioning, the reduction in latency are the major advantages of MOCHA architecture. The limitations include the speed, its complex architecture and it cannot be used in offline mode. Compared to MOCHA, VM Based Cloudlet architecture has improved speed, simple structure, reduced latency and improved QoS. But this architecture also has its own limitations. It uses more resources and will not work in offline mode. People should depend on the service providers for the cloudlet infrastructure to be deployed.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 6, Issue 8, August 2016)

Framework	Objective	Level of offloading	Implementation	Protocol
MOCHA	Minimize overall response time	Program partitions as per QoS	Using cloudlet with the capability of massively parallel processing using GPUs	WiFi/3G/Blue tooth
VM Based Cloudlets	Crisp interactive response by reducing latency, jitter, congestion and failures	Overlay	Using Hardware VM Technology	WiFi
MAUI	Primary aim is saving energy, speed is also considered	Method-RPC like	.Net Framework	WiFi/3G
CloneCloud	User Specifiable; reduction in execution time, energy usage, Transparent code migration	Method-Thread suspend and resume	Java Dalvik VM	WiFi/3G
ThinkAir	Scalability, efficient on-demand resource allocation, ease of use	Method level, semi-automatic offloading	Remoteable Code Generator and the Customized NDK	WiFi/3G
COMET	Transparent code migration	Threads	Java Dalvik VM	WiFi/3G
Dynamic Performance optimization	reduction in execution time, energy usage	Agent based application partitions	Java Agent Development Environment	WiFi
Chroma	User Specifiable; reduction in execution time, energy usage and increasing fidelity	Using effective decomposition and executing portions of the application on different resources	Tactic based	WiFi
MACS	enable transparent computation offloading of elastic mobile applications	Program modules	Android	WiFi
SOME	reduce the computational cost of mobile devices	Function	Javascript	WiFi/4G/LTE / WCDM/
Odessa	Responsiveness and accuracy	Threads	Sprout	WiFi/3G
Cuckoo	Elegant computation offloading, maximize computation speed, minimize energy usage	Method	Ibis, Android	WiFi/3G/Blue tooth
Scavenger	Improve performance and energy saving	Functions	Python	WiFi
IC-Cloud	effective computation offloading in a mobile environment where Internet access to remote computation resources is of variable quality and even	Functions	Android	WiFi/3G

Table 1: Comparison of different offloading frameworks

Adhoc cloudlets can be formed dynamically in the home network so that all the devices in that network can share the resources [41]. Secondly, better performance can be achieved by dynamically partitioning the application. In applications where the mobile acts as thin client and the whole application is remotely executed on the cloudlet, dynamic partitioning can improve the performance.

MAUI provides a fine-grained automated code offloading based on annotations and thus enables mobile applications to reduce energy consumption and improve their performance. As per the experiments conducted, this approach works only on Microsoft Windows and it provides method level code offloading based on the .NET framework. So this method lacks scalability. CloneCloud approach dynamically partitions the application by static analysis of Java code. It offers method level offloading and executes on a cloned VM image which reduces CPU utilization on mobile devices. Even though power saving is possible in mobile devices by using this idea, the requirement of pre-processing on the client side can increase the cost of the mobile device as well as the cloud to device network traffic. ThinkAir provides method level semi-automatic offloading. Scalability and parallel execution are the advantages of ThinkAir. COMET provides a mechanism to offload multithreaded applications using distributed shared memory.

In Chroma the application has to be partitioned manually and the remote tasks are executed using preinstalled RPCs. The scheduler will schedule the surrogates to perform different tasks. So the mobility of the client devices are restricted. MACS was developed as a successor of MAUI and Cuckoo frameworks. The partitioning decision is done at runtime and it does extra profiling and resource monitoring of applications. As SOME operates on JavaScript platform, it is a reliable, concise and platform independent approach. In Odessa the make span and throughput are improved by using a greedy incremental approach. To make offloading decisions it does not require prior information regarding the performance of application or a set of feasible partitions. Scavenger is dual-profiling history based scheduling approach which gives good performance in known and unknown environments. The tasks which are to be executed remotely are annotated and the Scavenger library will take care of installation and invocation. So the rewriting needed is reduced to minimum. IC-Cloud allows developers to annotate the tasks to be offloaded using a library and then collects features of all these offloadable tasks. Then the execution time is predicted accurately by IC-Cloud using machine learning.

IV. Conclusion

This paper covers an extensive literature on various architectures for code offloading. Code Offloading has been widely considered for saving energy and increasing the responsiveness of mobile devices. The effectiveness of an offloading system is determined by its ability to determine where the execution of code (local or remote) represents less computational effort for the mobile, such that by deciding what, when, where, and how to offload correctly, the device obtains a benefit.

REFERENCES

[1] J. Flinn, S. Park and M. Satyanarayanan, "Balancing Performance, Energy, and Quality in Pervasive Computing," in Proceedings of 22nd IEEE International Conference on Distributed Computing Systems , 2002.

[2] Y.-Y. Su and J. Flinn, "Slingshot: deploying stateful services in wireless hotspots," in Proceedings of the 3rd international conference on Mobile systems, applications, and services, Seattle, WA, ACM, 2005.

[3] R. K. Balan, D. Gergle, M. Satyanarayanan and J. Herbsleb, "Simplifying Cyber Foraging for Mobile Devices," in Proceedings of the 5th international conference on Mobile systems, applications and services, 2007.

[4] M. D. Kristense and N. O. Bouvin, "Developing Cyber Foraging Applications for Portable Devices," in 7th IEEE Conference on Polymers and Adhesives in Microelectronics and Photonics and 2nd IEEE International Interdisciplinary Conference on Portable Information Devices, PORTABLE-POLYTRONIC, Garmish-Partenkirchen, 2008.

[5] J. Porras, O. Riva and M. D. Kristensen , "Dynamic Resource Management and Cyber Foraging," in Middleware for Network Eccentric and Mobile Applications, Springer Berlin Heidelberg, 2009, pp. 349-368.

[6] B.-G. Chun and P. Maniatis, "Augmented Smartphone Applications Through Clone Cloud Execution," in Proceedings of the 12th conference on Hot topics in operating systems, HotOS'09, 2009.

[7] S. Abolfazli, Z. Sanaei and A. Gani, "Mobile Cloud Computing: A Review on Smartphone Augmentation Approaches," in Proceedings of the 1st International Conference on Computing, Information Systems, and Communication, Singapore, 2012.

[8] M. Ali, "Green Cloud on the Horizon," in Cloud Computing, Springer Berlin Heidelberg, 2009, pp. 451-459.

[9] H. T. Dinh, C. Lee, D. Niyato and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," in Wireless Communications and Mobile Computing, John Wiley & Sons, Ltd., 2011, p. 1587–1611.

[10] M. Satyanarayanan, "Pervasive computing: vision and challenges," IEEE Personal Communications, vol. 8, no. 4, pp. 10-17, 2001.

[11] R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen and H.-I. Yang, "The case for cyber foraging," in Proceedings of the 10th workshop on ACM SIGOPS European workshop , 2002.

[12] S. Goyal and J. Carter, "A lightweight secure cyber foraging infrastructure for resource-constrained devices," in Proceedings of the Sixth IEEE Workshop on Mobile Computing Systems and Applications, 2004.

[13] G. A. Lewis, P. Lago and G. Procaccianti , "Architecture Strategies for Cyber-Foraging: Preliminary Results from a Systematic Literature Review," Lecture Notes in Computer Science, vol. 8627, pp. 154-169, 2014.

[14] E. Cuervo, A. Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra and Paramvir Bahl, "MAUI: Making Smartphones Last Longer with Code Offload," in Proceedings of the 8th international conference on Mobile systems, applications, and services,MobiSys '10, 2010.

[15] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik and A. Patti, "CloneCloud: elastic execution between mobile device and cloud," in Proceedings of the sixth conference on Computer systems, EuroSys '11, Salzburg, Austria, 2011.

[16] M. Shiraz, A. Gani, R. H. Khokhar and R. Buyya, "A Review on Distributed Application Processing Frameworks in Smart Mobile Devices for Mobile Cloud Computing," Communications Surveys & Tutorials, IEEE, vol. 15, no. 3, pp. 1294 - 1313, 2012.

[17] T. Soyata, R. Muraleedharan, C. Funai, M. Kwon and W. Heinzelman, "Cloud-Vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture," in IEEE Symposium on Computers and Communications (ISCC), 2012.

[18] T. Soyata, R. Muraleedharan, J. Langdon, C. Funai, S. Ames, M. Kwon and W. Heinzelman, "COMBAT: mobile-Cloud-based cOmpute/coMmunications infrastructure for BATtlefield applications," in SPIE Proceedings -Modeling and Simulation for Defense Systems and Applications VII, 2012.

[19] M. Satyanarayanan, Bahl, P., Caceres R. and Davies N. , "The Case for VM-Based Cloudlets in Mobile Computing," Pervasive Computing, IEEE , vol. 8, no. 4, pp. 14-23, 2009.

[20] K. Ha, P. Pillai, W. Richter, Y. Abe and M. Satyanarayanan , "Just-in-time provisioning for cyber foraging," in Proceeding of the 11th annual international conference on Mobile systems, applications, and services-MobiSys '13, Taipei, Taiwan, 2013.

[21] J. Y., Tawalbeh L., Ababneh F. and Dosari F. , "Resource Efficient Mobile Computing Using Cloudlet Infrastructure," in IEEE Ninth International Conference on Mobile Ad-hoc and Sensor Networks (MSN), 2013.

[22] M. Satyanarayanan, G. Lewis, E. Morris,, S. Simanta, J. Boleng and K. Ha, "The Role of Cloudlets in Hostile Environments," IEEE Pervasive Computing, vol. 12, no. 4, pp. 40-49, 2013.

[23] C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul and C. Limpac, "Live migration of virtual machines," in Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation, 2005.

[24] A. Wolbach, J. Harkes, S. Chellappa and M. Satyanarayanan, "Transient customization of mobile computing infrastructure," in Proceedings of the First Workshop on Virtualization in Mobile Computing, MobiVirt '08 , 2008.

[25] "Tactical Cloudlets: Moving Cloud Computing to the Edge," [Online]. Available: https://www.sei.cmu.edu/mobilecomputing/research/tactical-cloudlets/.

[26] Y. Li and Wenye Wang , "The unheralded power of cloudlet computing in the vicinity of mobile devices," in IEEE Global Communications Conference (GLOBECOM), 2013.

[27] S. S., Lewis G.A., Morris E. , Kiryong Ha and Mahadev Satyanarayanan , "A Reference Architecture for Mobile Code Offload in Hostile Environments," in Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012.

[28] S. Osman, D. Subhraveti, G. Su and J. Nieh, "The Design and Implementation of Zap: A System for Migrating Computing Environments," in Proceedings of the 5th Symposium on Operating Systems Design and Implementation, Boston, Massachusetts, USA, 2002.

[29] S. Kosta, A. Aucinas, P. Hui, R. Mortier and X. Zhang, "ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in Proceedings of IEEE INFOCOM, 2012.

[30] M. S. Gordon, D. A. Jamshidi, S. Mahlke, Z. M. Mao and X. Chen, "COMET: code offload by migrating execution transparently," in Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation, OSDI'12, 2012.

[31] P. Angin and B. Bhargava, "An Agent-based Optimization Framework for Mobile-Cloud Computing," Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA), vol. 4, pp. 1-17, 2013.

[32] R. K. Balan, M. Satyanarayanan, S. Y. Park and T. Okoshi, "Tactics-based remote execution for mobile computing," in Proceedings of the 1st international conference on Mobile systems, applications and services, MobiSys '03, San Francisco, US, 2003.

[33] D. Kovachev and R. Klamma, "Framework for Computation Offloading in Mobile Cloud Computing," International Journal of Artificial Intelligence and Interactive Multimedia, vol. 1, no. 7, pp. 6-15, 2012.

[34] S. Park, Y. Choi, Q. Chen and H. Y. Yeom, "SOME: Selective Offloading for a Mobile Computing Environment," in Proceedings of IEEE International Conference on Cluster Computing (CLUSTER), 2012.

[35] M.-R. Ra, A. Sheth, L. Mummert, P. Pillai , D. Wetherall and R. Govindan, "Odessa: enabling interactive perception applications on mobile devices," in Proceedings of the 9th international conference on Mobile systems, applications, and services, MobiSys '11, 2011.

[36] R. Kemp, N. Palmer, T. Kielmann and H. Bal,"Cuckoo: a Computation Offloading Framework for Smartphones," in Mobile Computing, Applications, and Services, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 76, Springer Berlin Heidelberg, 2012, pp. 59-79.

[37] R. V. v. Nieuwpoort, J. Maassen, G. Wrzesińska, R. F. H. Hofman, C. J. H. Jacobs, T. Kielmann and H. E. Bal, "Ibis: a flexible and efficient Java-based Grid programming environment," Concurrency and Computation: Practice & Experience , vol. 17, no. 7-8, pp. 1079-1107, June 2005.

[38] K. M.D, "Scavenger: Transparent development of efficient cyber foraging applications," in Proceedings of the 2010 IEEE International Conference onPervasive Computing and Communications (PerCom) , Mannheim , 2010.

[39] C. Shi, P. Pandurangan, K. Ni, J. Yang, M. Ammar, M. Naik and E. Zegura, "IC-Cloud: Computation Offloading to an Intermittently-Connected Cloud," Georgia Institute of Technology, 2013.

[40] C. Shi, M. H. Ammar, E. W. Zegura and M. Naik, "Computing in cirrus clouds: the challenge of intermittent connectivity," in Proceedings of the first edition of the MCC workshop on Mobile cloud computing, MCC '12, Helsinki, 2012.

[41] T. Verbelen, Pieter Simoens, Filip De Turck and Bart Dhoedt, "Cloudlets: bringing the cloud to the mobile user," in Proceedings of the third ACM workshop on Mobile cloud computing and services, MCS '12 , 2012.

[42] Bahl, "cloudlets for mobile computing," 27 June 2014. [Online]. Available: http://research.microsoft.com/en-us/um/people/bahl/Present/WSC/Talk%203%20-%20Cloudlets.pdf.