Future Network and MobileSummit 2013 Conference Proceedings
Paul Cunningham and Miriam Cunningham (Eds.)
IIMC International Information Management Corporation, 2013
ISBN: 978-1-905824-36-6

Cloud-based Computation Offloading for Mobile Devices: State of the Art, Challenges and Opportunities

Lei JIAO[1], Roy FRIEDMAN[2], Xiaoming FU[1*]
, Stefano SECCI[3], Zbigniew SMOREDA[4] and Hannes TSCHOFENIG[5]
[1]*University of Göttingen, 37077 Göttingen, Germany*
{jiao,fu}@cs.uni-goettingen.de
[2]*Technion - Israel Institute of Technology, 32000 Haifa, Israel*
roy@cs.technion.ac.il
[3]*University Pierre and Marie Curie, 75005 Paris, France*
stefano.secci@upmc.fr
[4]*Orange Labs / SENSe, 92794 Issy les Moulineaux, France*
zbigniew.smoreda@orange.com
[5]*Nokia Siemens Networks, 02600 Espoo, Finland*
Hannes.Tschofenig@nsn.com
[*]*Corresponding author*

Abstract: Mobile cloud computing is a new rapidly growing field. In addition to the conventional fashion that mobile clients access cloud services as in the well-known client/server model, existing work has proposed to explore cloud functionalities in another perspective — offloading part of the mobile codes to the cloud for remote execution in order to optimize the application performance and energy efficiency of the mobile device. In this position paper, we investigate the state of the art of code offloading for mobile devices, highlight the significant challenges towards a more efficient cloud-based offloading framework, and also point out how existing technologies can provide us opportunities to facilitate the framework implementation.

1. Introduction

We are entering an era of cloud computing. Nowadays, people leverage cloud services from diverse aspects and enjoy various benefits of cloud computing. Cloud functionalities are exploited in many ways: Infrastructure-as-a-Service (IaaS), e.g., Amazon EC2 provides virtual machines that are used to deploy customized web services; Platform-as-a-service (PaaS), e.g., Google App Engine provides a comprehensive framework to build and deliver web applications; Software-as-a-Service (SaaS), e.g., e-mail services like Hotmail and web applications like Google Docs play an indispensable role in many people's daily lives. A key reason leading to an increasing commercial adoption of cloud computing is its unprecedented advantages over the conventional computing paradigm. Such advantages include, to name a few, reduced cost (e.g., users do not need to purchase infrastructures for infrequent computing tasks), easier maintenance (e.g., users do not need to install applications on their own client computers), and automatic scaling (e.g., the cloud can adjust the number of virtual machines to perform horizontal scaling according to the workload of the provided services).

People access public cloud services via the Internet. In turn, the Internet is increasingly accessed via mobile clients [1], due to the exploding prevalence of mobile handheld devices such as smart cell phones, tablet PCs and laptops. It is therefore not

surprising that recent market research predicts that by the end of 2014 the so-called "mobile cloud computing" market will deliver an annual revenue of about 20 billion US dollars [2]. Thus, it is fairly reasonable to foresee that mobile cloud computing will occupy a significant proportion of Internet usage in the future.

Despite the combined advantages of cloud computing with the ubiquitous accessibility of mobile devices, the full potential of mobile cloud computing is far from being fully exploited. For instance, the primary role of mobile devices today in the context of cloud computing is to serve as terminals for accessing the cloud. If we look at the interaction pattern between the cloud and the mobile device today, an application can be either executed on the cloud side and use the mobile side as a thin client, or executed on the mobile side alone, while potentially consuming isolated cloud services such as map services in GPS-based applications. Even so, what can be executed on the cloud depends on what concrete services are provided by the cloud. This status quo limits the possibility and flexibility to optimize the mobile application performance [3].

An intuitive idea for fulfilling this gap dates back to past experiences of distributed computing, which call for exploiting the cloud, as a remote execution environment, to improve the application performance via distributed execution between the cloud and the local device [4, 5]. When it comes to mobile handheld apparatus, this can be even more appealing since over the years the constrained computing and storage resources of mobile devices have remained a significant limiting factor of mobile devices. Moreover, mobile devices have serious power constraints due to their limited battery lifetime and the ever increase of richer (and more complicated, more computation-demanding) applications [6]. Furthermore, current battery technologies suggest that energy will remain the primary bottleneck in the foreseeable near future [7, 8]. In conclusion, there is a promising optimization space for distributed execution between the cloud and the mobile device.

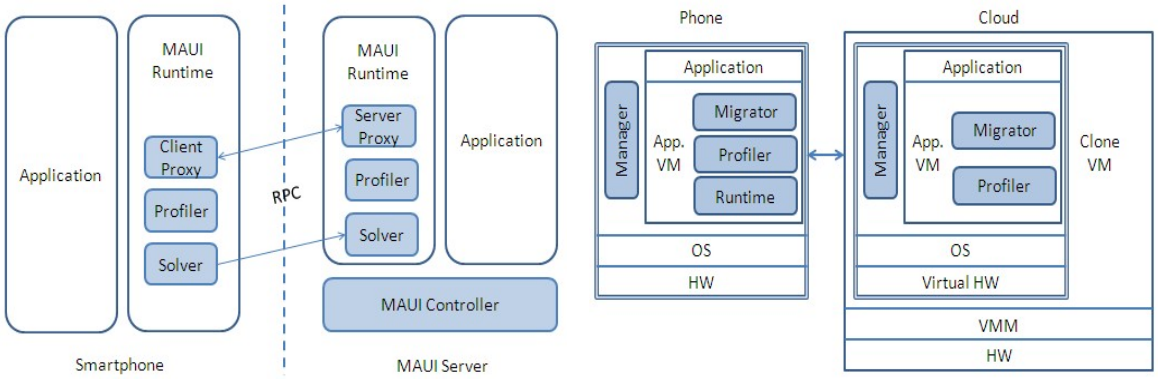
In this paper, we first investigate the state of the art of the distributed execution (i.e., computation/code offloading) between the cloud and mobile devices, and present a detailed comparison between these research efforts. Then we highlight the challenges towards a more efficient cloud-based offloading framework and also suggest some opportunities that may be exploited by future research work. Finally, we conclude our discussion with a brief summary.

2. State of the Art: Computation Offloading

Most related research efforts on computation offloading for mobile devices include MAUI [9] and CloneCloud [10]. In the following, we describe their principles from the application modeling to the system architecture, and then compare them in detail.

MAUI

MAUI offloads computation at the method granularity. It models an application as a call graph $G = (V, E)$. In the directed graph G, each vertex $u \in V$ represents a method and every edge $e = (u, v) \in E$ represents an invocation of method v by method u. The computation offload problem is converted to the graph partitioning problem so that one partition executes in the cloud and the other executes locally. The partitioning problem is further an integer linear program, in which the energy saving of the distributed execution is maximized, subject to the total execution time.



(a) MAUI (b) CloneCloud

Figure 1: State-of-the-art computation offloading architectures

maximize:

$$\sum_{v \in V} I_v \times E_v^l - \sum_{(u,v) \in E} |I_u - I_v| \times C_{u,v}$$

subject to:

$$\sum_{v \in V} ((1 - I_v) \times T_v^l + (I_v \times T_v^r)) - \sum_{(u,v) \in E} (|I_u - I_v| \times B_{u,v}) \leq L \quad (1)$$

$$0 \leq I_v \leq r_v, \forall v \in V \quad (2)$$

In the optimization objective, I_v is the binary decision variable for method v. $I_v = 0$ means v is executed locally while $I_v = 1$ means v is executed remotely. The above problem is to find the value of each decision variable for each method. E_v^l is the energy consumption of v if executed locally and $C_{u,v}$ is the energy spent on transferring the program state from the local device to the remote cloud when u calls v.

Constraint (1) indicates that the whole distributed execution time plus transferring time should be less than a predefined latency L. T_v^l is time cost of v if executed locally and T_v^r is the time cost of v if executed remotely. $B_{u,v}$ is the time spent on transferring the program state from the local device to the remote cloud when u calls v. In Constraint (2), r_v is the binary indicative variable, which means v should be considered as an offload candidate if $r_v = 1$ and should not be considered if $r_v = 0$. MAUI requires the programmer to mark in the source code which method should be considered to offload and which should not.

Given an application, in order to build the call graph and solve the optimization problem, MAUI has to profile the device, application and network to obtain the values of variables E_v^l, T_v^l, T_v^r, $C_{u,v}$ and $B_{u,v}$. MAUI profiles the device at initialization time to collect samples of CPU utilization and the corresponding smartphone energy consumption to build a simple linear energy model. MAUI also profiles the application continuously to acquire the execution time (i.e., T_v^l and T_v^r), the number of required CPU cycles, and the program state transfer requirements for each method. Besides, MAUI profiles the network continuously to measure the average throughput of the network interface. Leveraging all these profiling, MAUI can calculate the needed values.

MAUI is implemented based on Microsoft .NET framework on the cloud side and .NET compact framework on the mobile side. In its system architecture, as shown in

Figure 1(a), the Profiler is responsible for profiling the device, application and network to obtain necessary information as stated above; the Solver is responsible for solving the optimization problem periodically when the application runs; the Proxy takes care of the data and control transfer between the cloud and the smartphone.

CloneCloud

CloneCloud also offloads mobile codes at the method granularity. Two profile trees are built by the CloneCloud Profiler for each execution of the application: one for execution on the cloud and one for execution on the mobile device. Each node in the tree represents a method invocation and each edge represents a method call from the caller to the callee. Such representation is quite similar to MAUI's call graph. Besides, CloneCloud has a similar optimization model as follows.
maximize:

$$C(E) = Comp(E) + Migr(E)$$
$$Comp(E) = \sum_{i \in E,m} ((1 - L(m))I(i,m)C_c(i,0) + L(m)I(i,m)C_c(i,1))$$
$$Migr(E) = R(m)I(i,m)C_s(i)$$

subject to:

$$L(m) = 0, \forall m \in V_{Mobile} \tag{3}$$

$$L(m_1) = L(m_2), \forall m_1, m_2, C : m_1, m_2 \in V_{Native_c} \tag{4}$$

$$R(m_2) = 0, \forall m_1, m_2 : TC(m_1, m_2) = 1 \wedge R(m_1) = 1 \tag{5}$$

$$L(m_1) \neq L(m_2), \forall m_1, m_2 : DC(m_1, m_2) = 1 \wedge R(m_2) = 1 \tag{6}$$

The optimization objective is to minimize the cost, which can be the execution time (including the state transfer time to the cloud) or the energy consumption by the application on the mobile device. The cost is further divided into computation cost and migration cost. $R(m)$ is the binary decision variable for method m. $R(m) = 1$ means one migrates the method to the cloud while $R(m) = 0$ means no migration. $L(m)$ is the auxiliary decision variable which indicates the execution location of m: $L(m) = 1$ for remote execution and $L(m) = 0$ for local execution. $I(i,m)$ represents i is an invocation of m, which equals 1 if m is invoked and 0 if not. $C_s(i)$ is the migration cost of invocation i and $C_c(i, L(m))$ is the computation cost of invocation i at location $L(m)$. On a whole, the optimization problem is to find the value of $R(m)$ for each method m.

Constraint (3) indicates that methods accessing device-specific features must be pinned at the mobile device. Constraint (4) shows methods sharing the native states must be co-located. Constraint (5) means if one decides to migrate method m_1 to the cloud and m_1 calls m_2, then it must be located on the local device, i.e., to prevent nested migration. Constraint (6) means if one migrates a method, it will not be co-located with its caller .

A major advantage of CloneCloud over MAUI is that the former does not require source code availability and programmer efforts to identify the offload candidates, as what is done by the latter. CloneCloud adopts static program analysis to identify legal positions to put migration and re-integration points in the byte codes. CloneCloud is implemented based on Java Dalvik VM [11] and has the following architecture as shown in Figure 1(b). The components have similar functionalities to MAUI counterparts.

MAUI vs. CloneCloud

Essentially, MAUI and CloneCloud attempt to solve the same problem. The two proposals have similar models and architectures as well, although they differ substantially in implementation details. A detailed comparison is presented in Table 1.

Table 1: MAUI vs. CloneCloud

	MAUI	CloneCloud
Execution Prerequisites	Application binaries must be in both the mobile and MAUI-server sides	Device clone must be in the cloud; App binaries and partition database must be in the mobile device
Migration granularity	Method (RPC-like)	Method (Thread suspend and resume)
State transferred	Method level (referenced objects, variables, static classes, etc.)	Thread level (virtual state, program counter, registers, stack, heap etc)
Source availability	Required	Not required
Selection of migration candidates	Marked by developer as "Remoteable"	Identified automatically by static analysis
Optimization target	Maximize energy saving	Minimize execution time or energy consumption
Migration constraints	No UI codes; No local I/O involved; No methods that can be affected by re-execution; total execution time less than some pre-defined value	No mobile-specific methods; No methods that share native state(s); No nested migration
Execution prerequisites	Application binaries must be in both mobile and server sides	Device clone must be in the cloud; App binaries and partition database must be in the mobile device
Application model	Annotated call graph	Annotated profile tree
Energy model	Proportional to the number of CPU cycles	Calculated as power level multiplied by execution time. Map the triple to power level: <CPU(processing/idle), display(on/off), Net(on/idle)>
When to profile	Continuously at run time. No persistent profiling result is stored across multiple runs	Before the partitioned app begins. Profiling results are used to generate partition configuration files
When to solve	Periodically at run time	Before the partitioned app begins
Implementation	Based on .NET (compact) framework	Based on Java Dalvik VM

3. Towards More Efficient Computation Offloading

While MAUI and CloneCloud represent an important step towards the optimization of mobile application performance with the help of remote execution, major challenges towards a more efficient cloud-based computation offloading framework remain, as detailed below. We also explore possible alternatives that can leverage existing technologies and facilitate the implementation work.

Application Model

MAUI models an application as a call graph and CloneCloud uses profile trees. There is also some other work that models an application as a resource consumption graph [3]. Based on these models, the computation offloading problem is converted to a graph partitioning problem and/or further formalized to an integer linear programming problem. However, such approaches are limited in that they cannot properly and precisely reflect the real case of how complicated applications execute; e.g., such models assume applications are single-threaded and do not consider asynchronous method invocations. Multi-threading and asynchrony are usually necessary aspects of a huge number of modern mobile applications. To this end, novel application models are needed; one also needs to investigate what optimization problems the new models can result in and how this can impact the system architecture and implementation details. Although this requirement arises from the scenario of computation offloading, the application model can be an independent research issue which may be applicable for other scenarios.

Estimating Code Energy

MAUI profiles the mobile device and builds a simple linear model to estimate the energy consumption of a piece of code based on the number of CPU cycles this code requires to execute. CloneCloud calculates the energy consumption as a power level multiplied by the execution time of the code. Power levels are a series of pre-measured values,

and a given execution environment is mapped to a power value. The execution environment is represented by the triplet ¡CPU(Processing/Idle), Display(On/Off), Network(Transmitting/Receiving/Idle)¿. Both energy estimation models are inaccurate, because they do not consider some real-world factors that can usually largely impact the energy consumption and can result in wrong offloading decisions. These factors include, for example: (1) The energy consumption of I/O operations, which MAUI's and CloneCloud's energy models do not take into account; (2) The auto-scaling CPU clock speeds, one of the important modern CPU technologies, alters the CPU frequency automatically and therefore changes the execution time of a given code. It may be worth to adapt the energy management mechanisms for the mobile device from an OS perspective for a better energy estimation model for computation offloading [12, 13].

Exploring Cloud

Both MAUI and CloneCloud claim to be able to offload computation/execution from a mobile device to the cloud. However, their actual implementation and evaluation is done between a mobile device and a nearby PC/server, and does not exploit the cloud-side architecture, constraints and possible optimization opportunities when a number of mobile clients are simultaneously offloading codes. For instance, if some piece of code is frequently offloaded by multiple users, the cloud may reuse the virtual machine instances that hold this offloaded code without frequently allocating new and destroying existing virtual machines. This is somewhat similar to the notion of shared libraries in operating systems, as well as de-duplication in storage systems; both are known to result in significant savings. The cloud can also classify the codes into different categories to provide customized offloading service. Besides, due to the large number of potential offloading usages, the cloud may exploit the use of data mining and pre-profile some codes for further optimizing the offload process.

Both MAUI and CloneCloud are based on the assumption that the only optimization objective is to lower the energy consumption or improve the application performance of the mobile device, without taking into account the resources consumed at the offloading destination. However, the cloud provider providing virtual machines for offloading purpose will not only target at providing satisfactory user experience, but also try to achieve these goals while incurring the least cost in the cloud, e.g., providing customized offloading service with least resource (e.g., energy) consumption of the cloud itself. Some existing work has already discussed approaches to save energy in the data center [14–17] and in the cloud [18–21]. Other work models the energy consumption of the whole cloud computing system, i.e., the energy consumption from the client to the virtual machine as a whole [22], rather than only considering the client or the cloud alone. All these works provide thoughtful insights on making better offloading decisions based on a comprehensive optimization of the whole cloud computing system.

Ad-Hoc Cloud Offloading

As discovered in the MAUI work [9], the energy cost of computation offloading greatly depends on the round trip delay to the server. In particular, offloading over WiFi to a nearby machine is much more energy efficient than offloading to a remote Internet server over WiFi, which in turn consumes much less energy that using the 3G network. This motivates the looking at offloading computations to nearby devices, whether these devices are computers, laptops, tablets, or PCs, that are detected in an ad-hoc manner.

Such devices may happen to be on the same wireless LAN, connected through WiFi-Direct, or happen to be on the same WiFi ad-hoc network.

Facilitating this concept requires a discovery mechanism, such as Apple's bonjour protocol, as well as a distributed runtime mechanism that manages the offloading. A unique challenge in this space is the need to be able to recover from disconnection of devices. That is, the phone that owns the computation might wonder far apart from the device hosting its computation to the point that they can no longer communicate. As a result, the phone might need to restart the corresponding computation locally, or on another nearby device after realizing the disconnection. Yet, correctness should be ensured even in cases where the disconnection was short-lived, and the devices got reconnected again, despite the fact that the same computation might have been executed on more than one device. At the other side, if a device hosting a computation gets disconnected from the phone that initiated the computation, there is a risk of having an orphan computation that consumes resources but whose results will never be consumed. Even worse, to handle resources locked by the offloaded computation on the disconnected hosting device, all locks should be lease-based. Finally, the distributed runtime system should be able to choose which device a computation should be offloaded to. This should be done in a local and efficient manner.

Assisted Cloud Offloading in Mobile Access

Another way to reduce the round-trip time that would be encountered when offloading to an Internet server is to move the server close to user access points to decrease the access latency to acceptable values with respect to application performance. In fact, cloud servers are becoming mobile, and as users change point of attachment virtual server machines can also change their point of attachment, i.e., their cloud container. The virtual machine can therefore be migrated from a datacenter to mini-datacenters located close to the users to support efficient offloading. These mini-datacenters can be managed by the users themselves (e.g., at their own set-up-box at home), or by the mobile backhauling provider (e.g., when the application is a service of the mobile network provider itself), or by cloud surrogate in the access network managed in a similar way external Content Delivery Network surrogates are managed today in ISPs. An important challenge here is to guarantee IP continuity in the access to the offloading virtual machine (VM) being moved across mobile access cloud containers. In this scope, protocols like LISP can allow keeping the same IP address for the offloading VM. An important work in this direction is to optimize LISP to decrease the VM handover downtime on the path traced by [23] or acting directly in the data-plane, supposing the mobile equipment is itself a LISP mobile node [24].

As above mentioned, the offloading server may be shared among many users. An important aspect in this situation is to move an offloading server, shared by a group of users, to the cloud surrogate closer to the users, somewhere around the centroid of their different positions. Preliminary studies [25] show content consumption patterns are different at different periods depending on the presence of special events, with different grouping patterns among individuals. Using the huge amount of data available for analysis in nowadays mobile cellular and WiFi networks, it is also conceivable to predict future movements of users accessing the same services, having the same application usages patterns, i.e., potentially able to share the same offloading VM, so as to instantiate VM migration across mobile access VM containers.

Programming Model

MAUI and CloneCloud are implemented based on C# and Java, respectively. Both languages are imperative and object-oriented programming languages, which also support managed code execution environments, however suffering from two drawbacks for the offloading purpose: (1) global/static public classes/variables and (2) shared states across methods/threads. Therefore,to enable the RPC-style remote execution, MAUI has to ship all objects/variables referenced by the offloaded code to the remote side. For CloneCloud, methods sharing common states need to be co-located. Thus, the offloading flexibility is limited. Functional programming languages seem to be good alternatives for overcoming the above constraints intrinsically [26, 27], since they generally allow no global variables and/or shared states, and require each function to be stateless and re-executable. This characteristics fits well in the context of computation offloading. Moreover, functional languages provide more reliability support because one always obtains the same result by re-execution. This can be exploited for fault-tolerance in bad network conditions as desired by MAUI. In case offloading is done similar to CloneCloud, a key issue is to determine which specific function can be offloaded, so that no program state is needed to be shipped and each offloaded function can be encapsulated and executed in a remote thread independently.

A recent study on the impact of mobile code distribution using JavaScript for standards development has been published in the IETF [28]. Over the last few years JavaScript has been the favorite technique for providing a standardized mobile code distribution mechanism for many application developers. The JavaScript APIs are continuously extended to bring JavaScript applications to a level comparable with native applications. HTML5, a bundle of the new HTML version combined with several JavaScript APIs, has received widespread acceptance by web developers as well as in the smartphone app developer community. However, JavaScript currently only supports mobile code distribution from the server to the client where it is then executed in a sandbox; distributing code from the client to the server is not supported. An application developer typically decides during the application design time about the worksplit between the server and the client-side, often influenced by the nature of the application itself. Fortunately, a developer typically has the freedom to change the JavaScript code at the server side and code changes are instantly propagated to connecting Web clients. One potential direction for further research is thus to study how JavaScript may be re-used while at the same time offering a more dynamic code offloading.

Leveraging Concurrent/Parallel Runtime Support

MAUI and CloneCloud have to deal with the details of packaging and shipping either the referenced objects/variables or the program states because there is no underlying runtime support of saving and transferring such information. However, this is not the case for concurrent and parallel programming languages [29, 30]. Therefore, it is natural to consider concurrent/parallel languages as candidates to implement computation offloading because their runtime environment already provides the mechanisms for gathering information, shipping states and controlling remote execution so that one does not have to re-do this work. For example, it may be interesting to modify OpenMP [31] or COMET [32] to allow parallel execution between a mobile device and the cloud, taking the energy consumption of mobile codes into consideration.

4. Conclusion

In this paper, we have investigated the potential of cloud-based computation offloading. We started with reviewing the previous research efforts of MAUI and CloneCloud, and demonstrated the detailed comparison between these two proposals. Then, we argued that there are still great challenges preventing us from a more efficient offloading framework. We also pointed out several existing technologies that can help facilitating the implementation task. Moreover, security raises more and more attention in the cloud computing field and is also an important aspect to be considered in the next steps. We believe that mobile cloud computing will be the next big step for the future evolution of Internet-based computing and it is therefore necessary to develop efficient approaches to provide good mobile application experiences for users. We believe the research topic is very important, and it is not, as of our knowledge, addressed in ongoing FP7 projects on mobile cloud, such as Mobile Cloud Networking FP7 project [33].

References

[1] M. Meeker, S. Devitt, and L. Wu, *Internet Trends*. Morgan Stanley, 2010.

[2] M. Beccue and D. Shey, *Mobile Cloud Computing*. ABI Research, 2009.

[3] I. Giurgiu, O. Riva, D. Juric, I. Krivulev, and G. Alonso, "Calling the cloud: Enabling mobile phones as interfaces to cloud applications," in *Middleware*, 2009.

[4] B. Chun and P. Maniatis, "Augmented smartphone applications through clone cloud execution," in *HotOS*, 2009.

[5] Y. Zhang, H. Liu, L. Jiao, and X. Fu, "To offload or not to offload: an efficient code partition algorithm for mobile cloud computing," in *IEEE CloudNet*, 2012.

[6] R. Friedman, A. Kogan, and Y. Krivolapov, "On power and throughput tradeoffs of WiFi and Bluetooth in smartphones," in *IEEE INFOCOM*, 2011.

[7] R. Powers, "Batteries for low power electronics," *Proceedings of the IEEE*, vol. 83, no. 4, pp. 687–693, 1995.

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

[9] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: making smartphones last longer with code offload," in *MobiSys*, 2010.

[10] B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *EuroSys*, 2011.

[11] "Dalvik virtual machine," http://developer.android.com/guide/basics/what-is-android.html.

[12] A. Roy, S. Rumble, R. Stutsman, P. Levis, D. Mazières, and N. Zeldovich, "Energy management in mobile devices with the cinder operating system," in *EuroSys*, 2011.

[13] A. Pathak, Y. Hu, M. Zhang, P. Bahl, and Y. Wang, "Fine-grained power modeling for smartphones using system call tracing," in *EuroSys*, 2011.

[14] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "Elastictree: saving energy in data center networks," in *NSDI*, 2010.

[15] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *INFOCOM*, 2010.

[16] F. Ahmad and T. Vijaykumar, "Joint optimization of idle and cooling power in data centers while maintaining response time," in *ASPLOS*, 2010.

[17] S. Pelley, D. Meisner, P. Zandevakili, T. Wenisch, and J. Underwood, "Power routing: dynamic power provisioning in the data center," in *ASPLOS*, 2010.

[18] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy aware consolidation for cloud computing," in *HotPower*, 2008.

[19] P. Graubner, M. Schmidt, and B. Freisleben, "Energy-efficient management of virtual machines in eucalyptus," in *IEEE CLOUD*, 2011.

[20] M. Cardosa, A. Singh, H. Pucha, and A. Chandra, "Exploiting spatio-temporal tradeoffs for energy-aware mapreduce in the cloud," in *IEEE CLOUD*, 2011.

[21] F. Moghaddam, M. Cheriet, and K. Nguyen, "Low carbon virtual private clouds," in *IEEE CLOUD*, 2011.

[22] J. Baliga, R. Ayre, K. Hinton, and R. Tucker, "Green cloud computing: Balancing energy in processing, storage, and transport," *Proceedings of the IEEE*, vol. 99, no. 1, pp. 149–167, 2011.

[23] P. Raad, G. Colombo, D. Phung, S. Secci, A. Cianfrani, P. Gallard, and G. Pujolle, "Achieving sub-second downtimes in Internet virtual machine live migrations with LISP," in *IEEE/IFIP IM*, 2013.

[24] "Lisp mobile node implementation," http://www.lispmob.org.

[25] S. Hoteit, S. Secci, G. Pujolle, Z. He, C. Ziemlicki, Z. Smoreda, and C. Ratti, "Content consumption cartography of the Paris urban region using cellular probe data," in *ACM CoNEXT URBANE*, 2012.

[26] P. Hudak, "Conception, evolution, and application of functional programming languages," *ACM Computing Surveys*, vol. 21, no. 3, pp. 359–411, 1989.

[27] S. Peyton Jones, *The implementation of functional programming languages (prentice-hall international series in computer science)*. Prentice Hall, 1987.

[28] H. Tschofenig, B. Aboba, J. Peterson, and D. McPherson, "Trends in web applications and the implications on standardization," Internet draft, May 2012.

[29] G. Taubenfeld, *Synchronization algorithms and concurrent programming*. Prentice Hall, 2006.

[30] L. Valiant, "A bridging model for parallel computation," *Communications of the ACM*, vol. 33, no. 8, pp. 103–111, 1990.

[31] R. Chandra, R. Menon, L. Dagum, D. Kohr, D. Maydan, and J. McDonald, *Parallel programming in OpenMP*. Morgan Kaufmann, 2000.

[32] M. S. G. amd D. Anoushe Jamshidi, S. Mahlke, Z. M. Mao, and X. Chen, "COMET: code offload by migrating execution transparently," in *USENIX OSDI*, 2012.

[33] "Mobile Cloud Networking FP7 project," https://www.mobile-cloud-networking.eu.