Context-awareness in Opportunistic Mobile Cloud Computing

Radu-Corneliu Marin, Radu-Ioan Ciobanu, Radu Pasea, Vlad Barosan, Mihail Costea, Ciprian Dobre
University Politehnica of Bucharest, Romania

II. Peer-to-Peer systems for mobile cloud

ABSTRACT

Smartphones have shaped the mobile computing community. Unfortunately, their power consumption overreaches the limits of current battery technology. Most solutions for energy efficiency turn towards offloading code from the mobile device into the cloud. Although mobile cloud computing inherits all the Cloud Computing advantages, it does not treat user mobility, the lack of connectivity, or the high cost of mobile network traffic. In this chapter, we introduce *mobile-to-mobile contextual offloading*, a novel collaboration concept for handheld devices which takes advantage of an adaptive contextual search algorithm for scheduling mobile code execution over smartphone communities, based on predicting the availability and mobility of nearby devices. We present the HYCCUPS framework, which implements the contextual offloading model in an on-the-fly opportunistic hybrid computing cloud. We emulate HYCCUPS based on real user traces and prove that it maximizes power saving, minimizes overall execution time, and preserves user experience.

INTRODUCTION

Nowadays, the ubiquity of mobile devices is no longer a futuristic figment, but has actually become a present day reality. Spanning from wireless sensors to intelligent handhelds, mobile devices are gradually interweaving themselves into the surrounding environment, thus starting to resemble what Mark Weiser called “technologies that disappear” (Weiser, 1991). The advent of smartphones has brought forth cutting edge technologies in mobile computing that are rivaling those normally found in traditional desktop systems - dual-core or even quad-core platforms with extensive memory and storage capacities. These impressive features are augmented by high-end sensors, such as accelerometers, digital magnetometers and many more. But probably the most attractive feature of all is given by the high speed and large bandwidth wireless radios embedded in said devices, such as WiFi, 3G, and, as of currently, 4G networks. Furthermore, mobile software has also seen a shift in perspective as the need for rich applications to be deployed on a plethora of device platforms is growing in urgency.

The continuous development and improvement of smartphone platforms have yielded a most important issue which seems to have eluded mobile Original Equipment Manufacturers (OEMs) and software engineers alike, namely extreme energy consumption. Such a concern could have been ignored or even gone unnoticed if it hadn't overlapped with another pressing matter: battery manufacturers are struggling with the physical limit of current technologies (Schlachter, 2012). These two concurrent factors are endangering the reputation of both mobile hardware and software providers, as customer dissatisfaction is increasing with each passing day. It seems that the excelling computing power and the myriad feature sets of smartphones are being dragged down by low battery life (which eventually leads to the low availability of the mobile device).

Energy efficiency has been addressed by the mobile computing community either in hardware, by bounding resource consumption, or in software, by energy-aware scheduling and allocation of said resources to applications. This involves complicated techniques that focus on individual hardware components: the CPU (Liang, Lai, and Chiou, 2010; Liang and Lai, 2010), the display (Anand, Thirugnanam, Sebastian, Kannan, Ananda, Chan, and Balan, 2010; Ye, Dobson, & McKeever, 2012) or network transfers (Agarwal, Chitnis, Dey, Jain, Navda, Padmanabhan, Ramjee, Schulma, & Spring, 2010; Balasubramanian, Balasubramanian, & Venkataramani, 2009; Schulman, Navda, Ramjee, Spring, Deshpande, Grunewald, Jain, & Padmanabhan, 2010). Although such techniques provide undeniable power reduction, their applicability proves to be limited as smartphone requirements grow even larger, increasing at *exponential* rates. A more recent approach to deal with mobile energy efficiency is represented by the mobile cloud computing (MCC) paradigm, according to which power-hungry tasks are offloaded from mobile devices to be executed in the cloud. Thus, handhelds only require thin clients for cloud-enhanced applications. The advantages of MCC are invaluable to mobile application development as they include: extending the battery lifetime by moving computation away from mobile devices, better usage of data storage capacity and of available processing power, and improving the reliability and availability of mobile applications. Furthermore, MCC also inherits the benefits of cloud computing (CC) as well: dynamic provisioning, scalability and the *pay-as-you-go* model. However, the advantages of mobile clouds must be weighed by the shortcomings of such solutions. First of all, most MCC applications rely heavily on the availability and steadfastness of large bandwidth mobile networks (3G/4G); unfortunately, due to the inherent mobility of users, there is no guarantee that such communication media is always available and, in its absence, mobile cloud applications are unreliable or, in worst-case scenarios, are rendered useless or even faulty. Furthermore, mobile networks tend to be more expensive both financially and energetically, given that 3G is more power hungry that WiFi (Cuervo et al, 2010). Energy efficiency has yet to be taken into serious consideration by most MCC solutions and, ironically, in attempting to reduce power consumption, mobile cloud applications end up exhausting battery life through network transfers towards clouds instead of choosing local computation. An additional concern in MCC is the lack of privacy when storing data on public clouds as recent advancements prove it is best to make use of private or hybrid solutions for storing data more securely than previously attempted (Robison, 2010). Last, but not least, MCC implies that mobile application developers must understand, learn and make use of cloud computing, which can in turn entail financial burdens that were not present until now (Kemp, Palmer, Kielmann, & Bal, 2012).

In an attempt to solve the above-mentioned issues, we propose a hybrid solution where mobile devices are both clients and resources in an ad-hoc opportunistic mobile cloud. By putting together their combined resources, handhelds are able to reduce their global energy needs by intelligently collaborating over opportunistic networks (ONs). Our solution is not intended as a replacement for MCC, but instead it attempts to complement mobile clouds in order to fill the aforementioned missing gaps. We introduce the concept of *mobile-to-mobile contextual offloading*, in which handhelds make use of a contextual search algorithm to schedule the remote execution of tasks in trusted smartphone communities based on predicting the availability and mobility of nearby devices. We present the Hybrid Contextual Cloud for Ubiquitous Platforms comprised of Smartphones (HYCCUPS), a framework that implements the above contextual offloading model. HYCCUPS is focused on energy efficiency and, as such, it aims at bringing cloud resources closer to the mobile devices in order to reduce network transfers. Moreover, it makes use of opportunistic communication channels as they are less power consuming in comparison to mobile networks. By doing so, it also solves the connectivity problems of MCC applications in the lack of support for the latter. Opportunistic networking also addresses the problem of privacy, as data will only be shared within a trusted community, namely the hybrid cloud. Furthermore, the burden on mobile application developers is eased, as offloading is hidden from them behind inter-process communication constructs and, of course, because the offloaded tasks are executed on other mobile devices using the same (local) implementation. The novelty of the solution is two-fold: (1) we introduce a novel mobile smart collaborative cloud model formed between collocated mobile devices capable of working together and sharing their resources in order to the reach a global optimization goal (energy efficiency) and (2) we prove the validity of our cloud concept, by proposing and evaluating concrete resource management and scheduling algorithms in realistic scenarios, based on the use of contextual data, showing the capability of our solution to lead to a significant reduction in energy consumption.

Preliminary versions of our work were published in Marin and Dobre (2013). This chapter presents more extensive work, by providing the rationale behind the offloading model and behind HYCCUPS' proposed architecture and design, together with advanced evaluation results. Furthermore, it describes several extensions over the previously presented proposal, and it also provides insights on the challenges we were faced with while carrying out contextual tracing experiments, with an emphasis on the social aspects of collecting data from users.

BACKGROUND

Mobile energy efficiency is an urgent challenge that the mobile computing community is currently facing. There are two current trends that are targeted at solving the aforementioned issue: throttling resources in order to reduce power consumption, and offloading tasks in order to take advantage of more powerful computing infrastructures (e.g. clouds). Although the former techniques have rendered interesting results, we consider them to be limited, as they ultimately lead to a decreased user experience. On the other hand, mobile cloud computing is a more feasible solution, as it takes full advantage of current wireless technologies and of existing cloud computing infrastructures in order to enhance the mobile user experience, by paving the way for the development of rich context-aware applications.

The main idea behind mobile cloud computing is to offload the execution of power-hungry tasks and to move the data pertaining to said computation from mobile devices onto clouds, with respect to the intrinsic mobility of mobile users and human behavioral patterns. Hoang, Lee, Niyato, and Wang (2012) point out the main advantages of using MCC: extending the battery lifetime by moving computation away from mobile devices, better usage of data storage capacity and of available processing power, and improving the reliability and availability of mobile applications. Furthermore, MCC inherits the benefits of cloud computing as well: dynamic provisioning, scalability, multitenancy, and ease of integration. These features of MCC enable mobile application developers to create a consistent user experience, regardless of the multifarious devices on the market, thus offering fairness and equality to all users (Kemp et al., 2012). The remainder of this section tackles the background work, as well as other solutions related or similar to HYCCUPS. Not only do we describe solutions that offload execution from smartphones onto traditional computing clouds, but also solutions that imply mobile devices as being explicit resources in the cloud.

In the *mobile-to-cloud offloading* model introduced by MCC, most challenges arise from partitioning the mobile application code into tasks that can be executed remotely and tasks that are bound to the device, based on the dependencies of each task. As such, MCC solutions can be categorized by partitioning technique into *static* and *dynamic*, and by offloading semantics into *explicit* and *implicit*. The following proposed solutions directly impact the burden of the application developers when integrating with the cloud, as each solution must be weighed in terms of the problem that needs to be solved. For example, *implicit* solutions might be preferred, as they imply less understanding of cloud computing. However, they are more likely to lead to programming errors, as developers do not fully understand how device dependencies should be handled when offloading a task (e.g. failing to send the device context along with the task). There is no perfect answer for application partitioning, although, as will be further seen, composing the above techniques usually leads to better solutions.

Zhang, Schiffman, Gibbs, Kunjithapathamm, and Jeong (2009) highlight the need for creating a secure elastic application model which should support partitioning into multiple autonomous components, called *weblets*. They propose a secure elastic runtime which presents authentication between remote and local components, authorization based on weblet access privileges in order to access sensitive data, and the existence and identification of a trusting computing base for deployment of the elastic application. Furthermore, Zhang, Kunjithapatham, Jeong, and Gibbs (2011) extend the previous work by adding a contextual component responsible for the decision of offloading onto the cloud based on device status (CPU load, battery level), performance measures of the application for quality of experience and, of course, user preferences. By doing so, the application model supports multiple running modes: power-saving mode, high speed mode, low cost mode or offline mode.

As opposed to using explicit language constructs for remote code (Zhang et al., 2009; Zhang et al., 2011), Cuervo et al. (2010) use meta-programming and reflection to identify remotable methods in applications running on their system, called MAUI. The basic idea behind MAUI is to enable a programming environment in which developers partition application methods statically, by annotating which methods can be offloaded for remote execution. Furthermore, each method of an application is instrumented as to determine the cost of offloading it. The MAUI runtime then decides which tasks should be remotely executed, driven by an optimization engine which is aimed at achieving the best possible energy efficiency through the analysis of three factors: the device's energy consumption characteristics, the application's runtime characteristics, and the network characteristics. Moreover, they rule in favor of bringing cloud resources closer to mobile devices, as they prove that minimizing latency not only fixes the overwhelming energy consumption of networking, but also addresses the mobility of users. Chun, Ihm, Maniatis, Naik, and Patti (2011) also use static partitioning and dynamic optimization for offloading, but take such systems one step further by migrating the entire thread of execution into a virtual machine clone running on the cloud. This approach proposes maintaining a clone of the entire virtual machine from the mobile device onto the cloud, by constantly synchronizing the state of the device with its cloud counterpart, thus guaranteeing that an offload will execute correctly in terms of device context. Such an approach eases the mobile application developer's effort of assuring that all of an offloaded task's state and dependencies are correctly sent to the cloud. Moreover, given that the cloud will execute the same implementation (as if it were running locally), cloud computing knowledge is no longer a requirement, thus reducing the development life cycle of cloud-enhanced mobile applications. However, maintaining such a strict synchronization between the mobile device and the cloud incurs considerable mobile network traffic, which leads to high operational costs.

As opposed to the previous research, Chun and Maniatis (2010) rule in favor of dynamically partitioning mobile applications as to provide better user experience. The decision to execute remote computation is taken at runtime based on predictions obtained after offline analysis in the sense of optimizing what is actually needed: battery usage, CPU load, operational cost.

However, the benefits of using such solutions must be weighed with the security threats of cloud computing. The main concern of MCC is the lack of privacy in storing data on clouds as it faces multiple impediments: judicial, legislative and societal (Robison, 2010). Bisong and Rahman (2011) advise developers in MCC not to store any data on public clouds, but to turn their attention towards internal or hybrid clouds. In this sense, Marinelli (2009) proposes the use of smartphones as the resources in the cloud, instead of as its mere clients. He introduces Hyrax, a platform derived from Hadoop which offers cloud computing for Android devices, allowing client applications to access data and offload execution on heterogeneous networks composed of smartphones. Interestingly enough, Hyrax offers such features in a distributed and transparent manner towards the developer. Although Hyrax is not primarily oriented at reducing energy consumption, but at fully tapping into distributed multimedia and sensor data without large network transfers, it represents a valuable endeavor due to its general architecture, in which smartphones are the main resources offered by the cloud, while mobility is treated as a problem of fault tolerance.

Moreover, Huerta-Canepa, and Lee (2010) introduce a framework that allows creating ad-hoc cloud computing providers based on devices in the nearby vicinity which are focused on reaching a common goal, in order to treat disconnections from the cloud in a more elegant fashion. As such, they rule in favor of enforcing existing cloud APIs over communities of mobile devices as to allow seamless integration with cloud infrastructures, while preserving the same interface for the collocated virtual computing providers. On the other hand, Kemp et al. (2012) take advantage of the existing Android partitioning of the user interface (through Activities) from the background computational code (through Services) and inject remote execution of methods by bypassing the compilation process and rewriting the interface between the aforementioned components transparently to both the developer and the user. By doing so, they ease the design and implementation of MCC applications, as mobile application developers do not require any cloud computing knowledge, such as integrating with offloading APIs.

Murray, Yoneki, Crowcroft, and Hand (2010) take one further step towards opportunistic computing by introducing crowd computing. It combines mobile devices and social interactions, in order to take advantage of the substantial aggregate bandwidth and processing power from users in opportunistic networks, with the goal of achieving large-scale distributed computing. By deploying a task farming computing model similar to that of Marinelli (2009) onto real-world traces, they place an upper-bound on the performance of opportunistically executing computationally-intensive tasks and obtain a 40% level of useful parallelism.

Unlike the previous solutions, HYCCUPS proposes the creation of a hybrid computing cloud in which the main resources are smartphones, and focuses on reducing the overall consumed power of the interconnected devices. The main differentiating factor of HYCCUPS from other platforms is that the decision of whether to run a task remotely or locally is taken by a contextual search method based on a real-life trace model. Moreover, the context takes into consideration both the mobility and availability of a node to offload a task, and it also does not neglect the overall quality of experience of the users which are part of the cloud. Furthermore, similar to Kemp et al. (2012), we rule in favor of injecting the offloading code transparently to the developer, by hiding it behind inter-process communication constructs. By so doing, we preserve the mobile application development process, as the developer is unaware of the underlying mechanisms and is only left with the possibility of giving hints about code that may be offloaded. Such a development model encourages the decoupling of components and the modularity of mobile applications.

CONTEXT SENSING

In any mobile contextual search solutions, context is of paramount importance in imposing two necessary characteristics of fault tolerant and highly available mobile applications: proactiveness and adaptability (Hong, Suh, and Kim, 2009). If the former relates to reacting correctly to external stimuli, the latter refers to adjusting the behavior of the application upon context changes. Sensing and collecting contextual information thus becomes a cornerstone of situation-aware systems, as it is needed both internally (for adapting to environmental changes at runtime), but also externally (for offline processing and analysis).

In the past few years, there has been a growing interest for real-life user contextual data in the field of human dynamics, starting with the work of Barabasi (2005), which disproved the use of synthetic mobility models, and further introduced a queuing model that explained the heavy bursts and long tails in the distributions of events in human activities. More recently, Song, Blum, and Barabasi (2010) demonstrate once again the predictability of human behavior and mobility, by analyzing various user traces obtained from mobile carriers.

As for smartphones, collecting contextual data has proven to be an invaluable source for improving and even enhancing user experience. OEMs are deploying context tracing applications able to adapt to contextual changes, in order to reduce energy consumption and improve mobile data traffic. One of the most interesting approaches to sensing context is the *infamous* case of Carrier IQ (Holly, 2014), where OEMs and mobile telephony carriers were deploying a context tracing application software, IQ Agent, with the purpose of gathering, storing and forwarding diagnostic measurements on their behalf. The traced data included: battery levels, the state of the network, firmware, performance metrics for applications and web browsing, voice and data connectivity between smartphones and cell towers, and many more. This information was collected by an application running in the background, and each customer of Carrier IQ would decide which criteria was to be stored and forwarded to their servers for offline analysis. However, as opposed to performance metrics, it was discovered that Carrier IQ software was also tracking privacy infringing information, which led to a series of lawsuits in both civil and criminal court (Peralta, 2011).

In our case, collecting contextual data from real-world mobile users has multiple purposes:

1. Validating the feasibility of a solution such as *mobile-to-mobile contextual offloading* by proving the predictability of users interacting over wireless networks, as well as showing that the users naturally tend to form well-structured and durable communities over opportunistic networks.
2. Creating predictive models for availability and mobility to be used in contextual search algorithms.
3. Simulating our contextual search algorithm based on user-collected tracing data, in order to correctly measure the performance of our offloading model in real-life situations.

As such, in Marin, Dobre, and Xhafa (2012) we presented implementation details for an Android application which collects contextual data from smartphones, namely the **HYCCUPS Tracer**. The application is designed to run in the background and trace multiple features (see below), which can further be classified by the temporality of acquisition into static or dynamic, or by the semantic interpretation into availability or mobility features. Moreover, static properties are determined at application start-up and are comprised of the device's traits, while dynamic features are momentary values acquired on-demand. In terms of semantic interpretation, availability features represent values pertaining to the overall computing system state, while mobility features describe the interaction of the device with the outside world. The features collected by the HYCCUPS Tracer are as follows:

- Minimum and maximum frequency: static properties describing the bounds for Dynamic Voltage/Frequency Scaling (DVFS);
- Current frequency: momentary value of the frequency according to DVFS;
- CPU Load: the current CPU load computed from */proc/stat*;
- Total memory: static property of the device describing the total amount of memory;
- Available memory: momentary value which represents the amount of free memory on the device (bear in mind that, in Android, free memory is wasted memory);
- Out of memory: asynchronous event notifying that the available memory has reached the minimal threshold and, in consequence, the Out Of Memory (OOM) Killer will engage in stopping applications;

- Memory threshold: the minimal memory threshold that, when reached, triggers the OOM events;
- Sensor availability: static properties which convey the presence of certain sensors (e.g. accelerometer, proximity);
- Accelerometer: the accelerometer modulus is a mobility feature which characterizes fine grain movement (if available);
- Proximity: proximity sensor readings (if available);
- Battery state: the current charging level (expressed in %) and also the current charge state;
- Application activity: availability events representing user actions that trigger opening/closing application activities;
- Bluetooth interactions: momentary beacons received from nearby paired or discoverable devices;
- WiFi interactions: interactions over WiFi modeled using the Alljoyn framework[1]. AllJoyn is an open-source peer-to-peer software development framework, developed by Qualcomm Connected Experiences, Inc., which offers the means to create ad-hoc, proximity-based, opportunistic inter-device communication. The true impact of AllJoyn is expressed through the ease of development of peer-to-peer networking applications provided by: common APIs for transparency over multiple operating systems, automatic management of connectivity, networking and security and, last but not least, optimization for embedded devices;
- WiFi scan results: periodic wireless access point scan results.

The tracing process is executed both periodically (with predefined time intervals for different monitored parameters), and asynchronously (the process is triggered with the occurrence of events such as WiFi interactions or user events).

The HYCCUPS Tracer was used to carry out a tracing experiment in the Faculty of Automatic Control and Computer Science of the University Politehnica of Bucharest, Romania, which lasted for 65 days (March-May 2012) (Marin et al., 2012). A total of 66 students and faculty members volunteered and were selected to participate, in order to have a wide range of study years and specializations covered.

One of the main challenges we faced during the experiment was the lack of conscientiousness of the volunteer participants; we obtained an average usage rate of the tracing application of about 20-30%. Interestingly enough, other such studies (Barabasi, 2005; Song et al., 2010) also suffered from similar problems, although they were analyzing contextual data collected directly from mobile carriers (similar to Carrier IQ); they obtained an information factor about 20%, most likely caused by devices being turned off or outside of the service of the targeted cell towers. Based on the work from Barabasi (2005) and Marin et al. (2012), in Marin, Dobre, and Xhafa (2014) we proposed a methodology for assessing the predictability of human interactions, to be used when analyzing tracing data. The methodology operates with a low information factor, by focusing on the nodes with higher degrees of conscientiousness, and by computing predictability using a modified version of a compression algorithm which is able to work with incomplete data. Based on this methodology, we proved that not only do users adhere to stable and durable social communities based on wireless interactions, but we also showed an undeniable predictability of users interacting amongst themselves, as well as with wireless access points. Although the proposed methodology can aid researchers in studying patterns in human behavior, it does not solve the lack of conscientiousness, but merely bypasses it.

[1] https://www.alljoyn.org/

In order to delve deeper into this problem, we conducted a survey amongst the volunteers which have participated in the experiment, so as to discover the underlying issues regarding the lack of conscientiousness. We uncovered that there are two types of issues that led to the poor utilization of the tracing application: technical aspects and social aspects. While the former are only linked to an increased power consumption induced by the tracing application itself, the latter are more complex. Similar to the case of Carrier IQ, the volunteers in our tracing experiment were reluctant to participate due to the following issues:

1. Invasion of privacy. Users are less likely to share contextual data that can be easily tracked back to them. Although the tracing application does not collect any personal data, participants were wary about uploading tracing data containing their synergic patterns. Moreover, some users suspected that the tracing application was collecting additional privacy-infringing information as well.
2. Lack of interest (or even lack of understanding) regarding the purpose of collecting contextual data. Unfortunately, users have limited knowledge about the possibilities/advantages of context-aware applications. Also, due to the fact that there is a limited set of situation-aware applications available on the market, mobile users have not had direct access to software that takes full advantage of the surrounding environment to augment their mobile experience.

In order to increase the degree of conscientiousness for future tracing experiments, we have redesigned the HYCCUPS Tracer so as to attempt to solve the above-mentioned issues. While tackling the reported increased power consumption caused by the tracing application, we have uncovered a design flaw that was forcing the CPUs to remain online instead of entering sleep states. Not only was this issue incurring additional overhead, it was also biasing the collected CPU tracing data. Furthermore, after analyzing the behavior of each implemented context collector, we noticed that the Bluetooth interactions tracing was the most power hungry task in the application. Fortunately, Marin et al. (2012) proved that WiFi opportunistic interactions are much more feasible than those over Bluetooth, as the latter tend to isolate micro-communities due to lower range, which eventually leads to far less synergy between nodes. As such, we were able to trim down this feature, leading to a much lower power consumption of the entire tracing infrastructure.

In terms of privacy, we have employed multiple measures to guarantee the transparency and anonymity of the tracing procedure. First of all, we have enforced a Model-View-Controller (MVC) architectural pattern for context collectors. By doing so, we are forcing the implementation of each context tracer to display the collected context to the user. Therefore, the suspicions that our application is secretly gathering privacy-infringing data should be reduced. Furthermore, we will be making the source code of the application available to all participants in future experiments, to further prove the soundness and transparency of our solution. Additionally, we are applying anonymization techniques, by computing and storing only the hash sums of all potential privacy-infringing information, such as wireless access points SSIDs and BSSIDs. This measure should solve the issue of contextual data being traced back to the originating device or user. Last, but not least, each user will be given secure access to her uploaded data for offline browsing.

While the first two issues have rather intuitive solutions, the last social aspect, namely the lack of interest or understanding about the purpose of contextual tracing, is more challenging, due to the fact that it is not related to the actual tracing application, but rather to how users perceive it. As such, we believe that a potential solution for growing user interest in context-aware applications is to provide incentive examples which demonstrate the importance and novelty of tracing and using context to enhance and augment the mobile user experience. Therefore, we rule in favor of developing such applications and deploying them in future tracing experiments alongside the HYCCUPS Tracer, in hope of increasing the conscientiousness of participants. The first step taken in this sense is to modify and extend the functionality of the tracing infrastructure so that it offers the collected contextual data to any application requesting access to it. By doing so, we are enabling mobile application developers to make use of the HYCCUPS Tracer as a means for sensing and sharing context.

Furthermore, we devised Chatty, an Android application for exchanging messages in opportunistic networks. It is implemented over the WiFi interactions collector, taking advantage of contacts between mobile nodes to carry messages from sources to destinations. Chatty allows two types of communication: point-to-point (where a node *A* wants to send a message to a node *B*) and dissemination (where a node *A* marks a message with certain tags, and sends it to all the nodes that are interested in receiving data marked with at least one of those tags).

Although Chatty is used for communication in opportunistic networks, where no Internet connection is assumed, the user is required to connect with her Facebook account when installing the application. This is done for two reasons, the first one being that the application needs a list of contacts for point-to-point communication. Thus, we decided to use the list of Facebook friends that have installed the Chatty application. Moreover, we consider a user's ID in the network to be the same as her Facebook ID, for the sake of simplicity. Secondly, the opportunistic communication module benefits from social network information when taking routing and disseminating decisions, in order to optimize its behavior in terms of hit rate, delivery latency and congestion. Once a user logs for the first time, there's no further need for Internet access, since the communication is done opportunistically. Aside from only offering chatting capabilities, Chatty also allows a user to see when one of her contacts is in opportunistic communication range, or (if not) when a contact was last seen in range. As an example, this can be useful in a crowded room, if a user is searching for a friend.

We envision Chatty as a helpful tool in an academic environment, to be used for communication between students, teachers, the faculty offices, etc. For instance, it can be used by professors or assistants to disseminate information about classes, or by the faculty offices to disseminate various announcements. In order to do the latter, a message is sent with a tag such as "faculty announcement", and all students that have this tag marked as an interest receive the message opportunistically, after a contact with a node that is a carrier for that message. Communication can also be performed between students located in opposite corners of the campus, through contacts with other mobile device owners that have installed Chatty. We consider a faculty campus as a suitable deployment environment for Chatty, since it is a small environment, with a high density, therefore the number of contacts is high.

We have conducted a tracing experiment on a smaller scale in a controlled academic environment by deploying the enhanced HYCCUPS Tracer along with the Chatty incentive application and we have observed a higher degree of conscientiousness after only a few weeks of tracing, with a usage rate of about 40-50%. Therefore, we are planning on developing a more complex set of incentive applications similar to Chatty, as to further carry out larger scale tracing experiments in the hope of collecting contextual data with a higher degree of information.

MOBILE-TO-MOBILE CONTEXTUAL OFFLOADING

In order to promote energy efficiency, we propose a novel mobile collaboration solution based on a contextual search algorithm which schedules remote execution of tasks in an opportunistic community by predicting the availability and mobility of nearby devices, namely *mobile-to-mobile contextual offloading*.

Contextual search is a technique used in pervasive systems to augment a query with context data from users in order to obtain content optimized for a specific situation (Kraft, Maghoul, and Chang, 2005;

Minkov, Cohen, and Ng, 2006; Challam, Gauch, and Chandramouli, 2007). In order to perform contextual search, multiple phases are required:

1. sensing contextual data: gathering or tracing raw context data from sensors in the environment;
2. aggregating and recording traced data: aggregating raw context data into higher level context and persistently storing it;
3. conducting the context-enhanced search: augmenting queries by means of the contextual records.

While sensing context has been thoroughly treated in the previous section, we are making use of the collected tracing data in the experiment conducted in Marin et al. (2012). Out of the myriad traced features, only the following are considered interesting for recording: CPU load, CPU frequency, available memory, interactions with peers over WiFi, battery level, battery charging state and application activity events. The tracing data is aggregated into the trained availability records; each record behaves similar to a circular buffer, as it contains aggregated contextual data for each day of the week with a given sampling rate. We consider a week to be sufficient, as human behavioral patterns in academic and office environments are subject to repeatability within this interval. When storing a feature value, it is aggregated with the previous tracing data at the closest sampling point smaller than the timestamp of the event. Based on the aggregation method, there are two types of records:

1. **Averaging records:** the data stored in these records is a continuous real value and it is aggregating by using the average;
2. **Probability records**: these records contain boolean events and the aggregation method is the probability that the event is true.

Based on the features of interest, Table 1 depicts the types of records used and the sampling intervals:

Feature	Record Type	Sampling interval (min)
CPU Load	Averaging	1
CPU Frequency	Averaging	1
Available Memory	Averaging	1
Battery Level	Averaging	5
Battery Charging?	Probability	10
Peers Connected?	Probability	10
User active?	Probability	10

Table 1: Record types and sampling intervals for features of interest

The sampling intervals were chosen by inspecting the inter-event times for each feature from the HYCCUPS tracing experiment in the previous section, and also by taking into consideration the mobile development aspect. Given that *mobile-to-mobile contextual offloading* will be deployed onto handhelds, the size of the records is important. Therefore, in the cases of CPU load and frequency, we made a compromise between informedness and spatial restraints. The reader should be advised that, although the CPU features will be less reliable, they will still offer valuable quantitative information.

As long as the contextual search is running on a device, the records are continuously updated so that the offloading algorithm is able to adapt to environmental changes (e.g. user changes her patterns or battery health degrades). However, these averaged momentary values have no meaning on their own, they simply represent an instance in a user's history. In order to make good use of these values, we need to extract temporal patterns over intervals of execution so that the system can predict future values for them.

In order to predict the values for features on any given interval of time, the offloading model uses univariate linear regression to determine a hypothesis h of the form $h(x) = \Theta_1 + \Theta_2 \times x$, which is able to approximate values for features for any given timeframe. So basically, the predictor for the average of a feature of over the interval $\left[currentTime, currentTime + \Delta t\right]$ is of the following form: $predictFeature(\Delta t)$ and it computes this future value by following the steps below:

1. Construct the statistical analyzed interval: $\left[currentTime - \Delta t, currentTime + 2\Delta t\right]$; we triple the predicted interval as to provide sufficient information for the actual prediction.
2. Extract the n samples from the feature's record in the statistical analyzed interval. The records are considered to be circular (the weeks wrap around).
3. Construct the X matrix, where x_i is the i-th sampling point in the analyzed interval:
$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \hline 1 & x_n \end{bmatrix}$$
4. Construct the y column vector, where y_i is the i-th feature value:
$$y = \begin{bmatrix} y_1 \\ y_2 \\ \hline y_n \end{bmatrix}$$
5. Compute the Θ column vector, by solving the following equation: $X \times \Theta = y$. The Θ column vector contains the parameters of our heuristic h which is able to predict values of a feature and is computed using the normal function:
$$\Theta = (X^T \times X)^{-1} \times X^T \times y$$
6. Compute the average of the feature in the prediction interval:
$$h(\frac{currentTime + \Delta t}{2})$$

Given that such an offloading model is constantly adapting to environmental changes, it is only natural to believe that predictors may output erroneous values. In this sense, the prediction output is weighed against degree of informedness and to their own accuracy:

$$predictFeature(\Delta t) = w \times h\left(\frac{currentTime + \Delta t}{2}\right) + (1 - w) \times momentary\_value$$

where w is the feedback weight, computed as:

$$w = \frac{informedness}{2} + \frac{accuracy}{2}$$

The degree of informedness expresses the number of samples that have been recorded in the statistical analysis interval:

$$informedness = \frac{no\_filled\_samples}{no\_total\_samples}$$

In order to measure the effectiveness of the predictor, our offloading model uses balanced accuracy:

$$accuracy = \frac{1}{2} \times \frac{positives}{positives + false\_positives} + \frac{1}{2} \times \frac{negatives}{negatives + false\_negatives}$$

The heart of the *mobile-to-mobile contextual offloading* model is the contextual search algorithm based on the above feature prediction model, which is responsible for deciding whether to offload a task on a nearby device or to execute it locally. To be able to schedule tasks in opportunistic networks, the contextual search is split into two phases:

1. Workload profiling: ascertaining the amount of resources needed to actually run a task.
2. Offloading decision: determining if a task is better run locally or on a different device.

Given that any task can run on multiple devices, each of them having specific traits, workload profiling needs to first determine which resources are required by the task (and to what extent), compute a workload score which uniquely describes the computational requirements of the task and, last but not least, normalize the workload score to be able to compare executional needs of a task for multifarious mobile platforms. In this sense, we introduce the computational potential (or simply potential), which is defined as follows: the computing potential is the time required by the processor to run a workload, considering current and predicted values for CPU load and CPU frequency in the nearby future.

In essence, the potential is a measure of the CPU time required by a task on a specific device. At a first glance, this measure may seem inequitable, as devices with better performing CPUs might seem disadvantaged, but (as we shall see further on in this section) time turns out to be a good invariant in our problem. Let us take on an example: two devices - a low end device (A) and a high end device (B) with a 5 times faster CPU - are currently interacting; device (A) shortly becomes overwhelmed by the tasks it is executing, so it offloads 5 tasks with $potential_A = 1s$ to (B), which evaluates each at $potential_B = 0.2s$ so, instead of (A) executing them in 5s, (B) runs all of them in 1s; not much later, the user of device (B) becomes active and launches a task which (B) evaluates to $potential_B = 1s$, and offloads it to (A), which evaluates it to $potential_A = 5s$. Now both devices have cleared their debt and were able to run more efficiently (while the remote device was executing the offloaded task, the local device was able to execute other tasks). The reader should bear to mind that the workload of a task is usually directly proportional to the computational possibilities of the device that it's running on (because users tend to use applications that are suited for their device). As such, tasks from low end devices such as (A) are usually much smaller than tasks running on high end devices such as (B).

Furthermore, in order to reduce the number of leechers in the system, a cost model was put in place. Each time a node offloads a task for another, it remembers the debt it is owed by that node as the sum of negative potentials; also, the node which is requesting the offload stores the debt it owes the other node as the sum of positive potentials. When a node responds to an offload request, it will advertise that it's potential is: $potential = true\_potential - debt\_for\_requestor$. This ensures altruism, as it discourages nodes to take advantage of other nodes, and it encourages nodes with debt to offload. in order to reduce their debt.

Having said that, it is high time we moved on to computing the potential of a task on a device. The following steps are required to determine the potential, given the number of CPU cycles estimated for a task through workload profiling:

1. Determine the maximum acceptable time (MAT) to run a task (the time needed to run a task at lowest frequency with 50% CPU load):

$$MAT = num\_cycles \times \frac{1}{0.5 \times min\_frequency}$$

2. Determine the potential of a task (the predicted time to run a task in current conditions):

$$potential = num\_cycles \times \frac{1}{predicutCPULoad(MAT) \times predictCPUFreq(MAT)}$$

Although it's not the absolute maximum time of running a task on a device, the maximum acceptable time represents the absolute maximum time that we would allow an offloaded task to run on a device, and it should be considered as the lower acceptable limit for offloading a task. It's more of a quantitative approach to determine the interval of time that a task should be executed in. As such, we compute the actual potential by predicting the CPU performance over the MAT interval. Therefore, the maximum acceptable time is computed only to determine the interval of prediction for the potential.

Based on a task's estimated potential, the offloading model is now able to correctly schedule a task in an opportunistic network. When a task is generated by an application, it passes through the workload profiler, which determines its computing potential. This potential will further augment the contextual search algorithm that actually decides whether to execute the task locally or offload it. Given that the task is executed in an opportunistic network, the decision is distributed. Therefore, there are actually two types of decisions that need to be taken:

1. **Local decision**: the current device decides whether it is best to execute the task locally (e.g. the device is charging) or whether it should be offloaded.
2. **Remote decision**: a remote node receives the offloading task request and it needs to determine if it can handle the execution of the remote task.

The local decision

Before offloading a task onto nearby devices, a node first decides, through contextual search, if the task actually needs to be offloaded and executed remotely, or if is better run locally. Figure 1 depicts the decision tree for local resolution of tasks. Each decision node in the tree is actually a feature predictor, as previously described in this section.

If the offloading model decides to remotely execute the task, it sends a task offload request to all connected peers, and waits for their potentials to run the task. After gathering all of the results, it adds the local potential to the list as well, and chooses the best candidate that will use minimal potential to execute the task. The reader should bear in mind that the task can still be run locally if the local node has the smallest potential. It should be noted that, under certain conditions (e.g. when the local device is charging), the offloading model is bypassed and the task is directly executed locally.

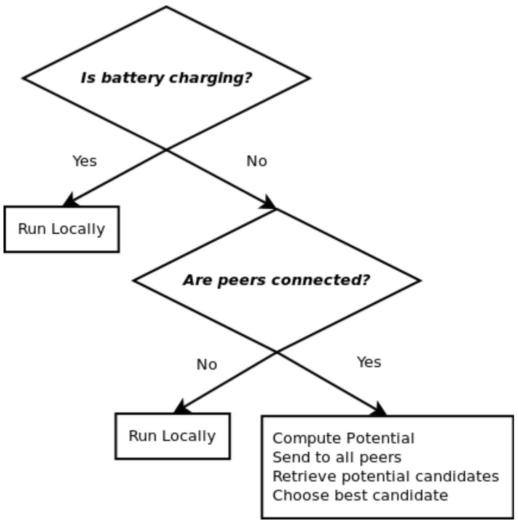

Figure 1: Local decision for offloading

The remote decision

When a node receives an offload request, the offloading model needs to decide whether it will accept to run the task, or if it will reject it because it is unable to spare the resources. The contextual search in the remote decision is illustrated in Figure 2. Similar to the local decision, each node in the decision tree is a feature predictor, as previously described in this section.

It is fairly obvious that the remote decision is much more intricate than the local one, because each node wants to maximize the number of offloaded tasks, while minimizing the offloading it's actually doing. However, the cost model presented above is refereeing this process so that no node will be taken advantage of by others.

Also, similar to the local decision, under certain conditions (e.g. when a node is charging), it will attempt

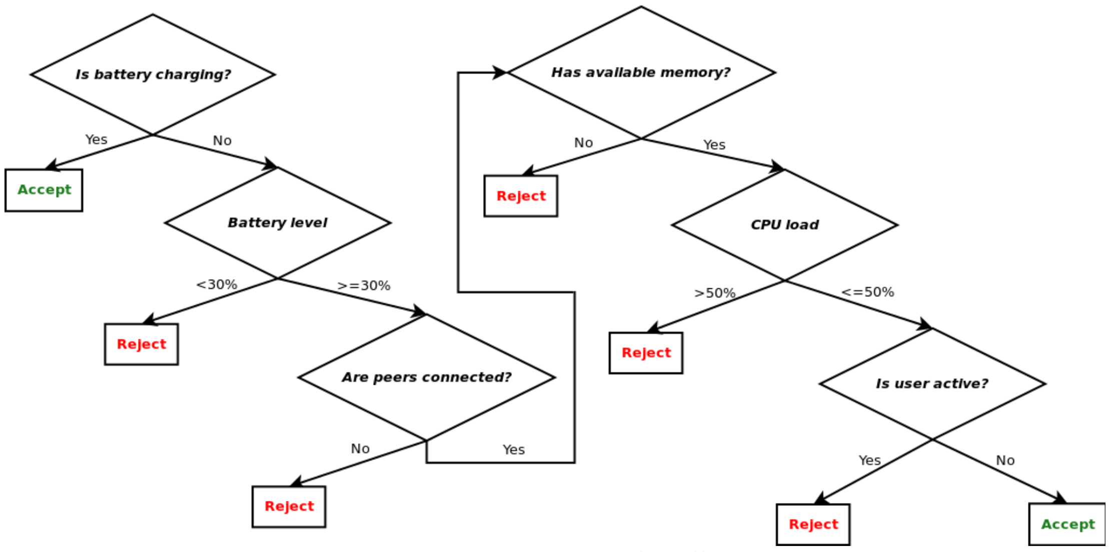

Figure 2: Remote decision for offloading

to act altruistically and it will advertise only a tenth of its actual potential to execute the task. The main reason behind this is to encourage other nodes to offload to it while it is still charging.

HYCCUPS

The conceptual framework

HYCCUPS is a ubiquitous computing conceptual framework which proposes to offer smartphone devices the opportunity to collaborate over high-speed wireless networks in a distributed and transparent manner, in order to reduce the overall power consumption by implementing the *mobile-to-mobile contextual offloading* model. The cornerstone of the solution stems from the use of smartphones as both the computing resources, but also as the clients in a hybrid cloud. As such, the framework was designed to fully cover the four issues stated by Marinelli (2009) in taking such an endeavor:

1. each node is owned by a different user;
2. each node is likely to be mobile;
3. the network topology is dynamic;
4. each mobile node is battery powered.

As will be seen in the remainder of this section, the issues enumerated above influence the entire design and deployment process. Furthermore, HYCCUPS attempts to solve the issues pointed out by Cuervo et al (2010) in the MAUI system:

1. WiFi should be used instead of mobile networks (3G), because it is more energy-efficient: HYCCUPS is designed to be deployed on WiFi networks as the cloud resources (the mobile devices) are discovered by using a bus over a wireless access point.
2. Cloud resources need to be closer to the mobile devices, as to reduce the network transfers: when using HYCCUPS, all cloud resources reside in the local network in close vicinity of each other.

In the HYCCUPS framework, partitioning of remotable code is done statically at task level by injecting the offloading code in language constructs specific to the smartphone platform on top of which it is deployed. Although this process is transparent to the developer, it is still her responsibility to give hints about which tasks can be remotely executed. We prefer to inject this implementation behind message sending subsystems, which allow communication between components of the same or even of different processes and applications. As such, we enforce modularity in the mobile application development process. Moreover, we create a certain type of symmetry, as the developer must write both the user interface, as well as the task executor module. As such, when the task is executed either remotely or locally, it is treated in the same manner by the same module implementation.

Considering that a mobile cloud is, by definition, always in perpetual motion, and that its components are usually owned by multiple users, a node's availability of processing a submitted workload is uncertain. As such, HYCCUPS uses a contextual search method to determine if a workload should run remotely or locally, and acts as a medium of offloading the current task to the best candidate in the mobile cloud, as to reduce the power consumption while preserving the quality of experience. Also considering the heterogeneity of a cloud, a node needs to be able to update its trained data by means of a feedback process, in order for it to evolve as to guarantee better availability resolution.

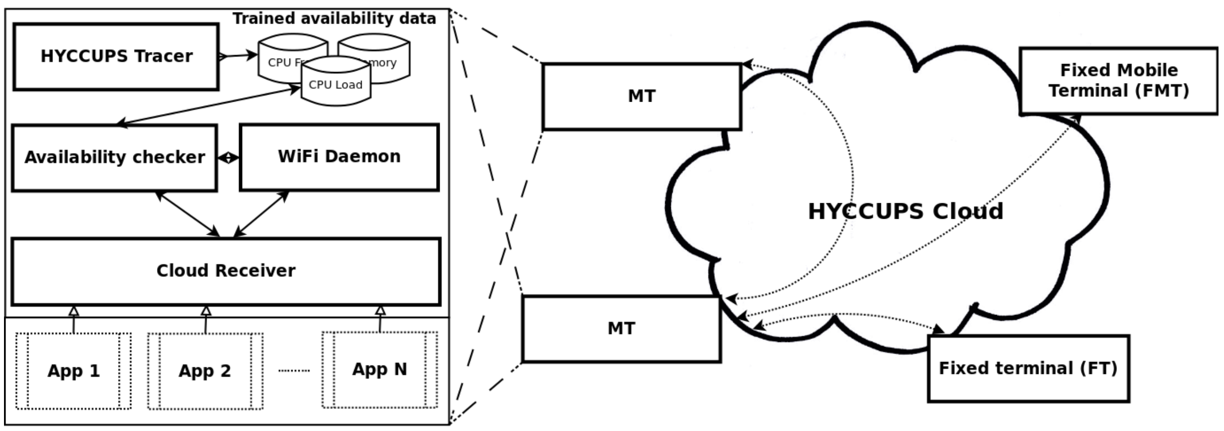

Figure 3: HYCCUPS architecture

The main purpose of the HYCCUPS framework is to provide a middleware layer on top of which developers will be able to seamlessly design, build and deploy applications which will take advantage of the full potential of contextual execution offloading. The overall architecture of HYCCUPS is illustrated in Figure 3. The right hand side of Figure 3 represents the hybrid cloud deployment over a WiFi network. There are three types of cloud terminals:

1. **Mobile Terminal (MT)**: a smartphone terminal, not plugged in. Its main characteristic is mobility. Not always available for executing workloads.
2. **Fixed Mobile Terminal (FMT)**: a plugged-in smartphone terminal. Acts as a Fixed Terminal, but may rapidly return to a mobility state (MT).
3. **Fixed Terminal (FT)**: a wearable computer. Will always be available for executing workloads. This is an optional component, as it raises compatibility issues: the developer has to explicitly redesign the mobile application code so it can be deployed on other platforms.

The left hand side of Figure 3 illustrates the anatomy of an MT (because FT and FMT do not impose any issues, the reader should expect them to act as regular cloud computing resources). The main components of an MT are:

1. **HYCCUPS Tracer**: as described in Section 3, this component is an application designed to collect contextual data from smartphones.
2. **Trained availability data records**: persistent datastores that aggregate the contextual information gathered by the HYCCUPS Tracer, also as described in Section 3.
3. **WiFi daemon**: the component in charge of creating and maintaining the WiFi bus, and also responsible for resource discovery. This component is the actual glue holding the entire cloud together.
4. **Availability Checker**: this component is the cornerstone of HYCCUPS' foundation, because it is responsible for conducting the contextual search that determines if a task should be offloaded and/or if an incoming remotable task should be executed locally.
5. **Cloud receiver**: the component responsible for resolving the execution runtime of incoming/outgoing remote tasks. If the decision of offloading is done by the Availability Checker, the actual enforcing of that decision is done by the Cloud receiver.

We rule in favor of tracing models, since we find that they better map onto real life situations than synthetic models, as Barabasi (2005) proves that the distributions of inter-event times in human activity are far from being normal, presenting bursts and heavy tails. It is for this reason that we use custom tracing applications to be able to acquire all needed mobility, availability and quality of experience tracing data. Moreover, the tracing interval can be set to reflect the particular needs of the mobile cloud users. For example, in academic and office environments the usual timing unit is an hour and is quite stable, whereas in a shopping mall users are quite mobile and the tracing interval should be far less (in the order of minutes).

We take the following features into consideration as tracing data:

- mobility: we are more interested in wireless mobility than geographical tracking; as such, we aim at tracing patterns of users interacting over WiFi;
- availability of the device: we collect statistical information about available resources such as CPU frequency, CPU load, available memory and battery statistics;
- quality of experience: we are interested in discovering when the user is performing operations on the device (opening/closing applications).

The machine learning algorithm in the Availability Checker is responsible for aggregating the above-mentioned tracing data into a predictor which is able, based on the timing interval, to determine that the device is available to offload a task, that the device is stable mobility-wise for a sufficient interval as to process the remote task, and that the owner of the remote device is not experiencing difficulties due to the executing task.

The basic behavior of an MT is to attempt to offload execution onto the cloud and respond to other workloads submitted into the cloud. There are two typical scenarios:

1. An application submits a workload onto the cloud. Based on the Availability Checkers' offloading decision, the Cloud Receiver passes the task either to the WiFi daemon, or to the local application. Needless to say, because of the middleware layer, the user will never know if the submitted workload was offloaded or if it actually executed locally. The mobile cloud does not impose that all execution be done on other devices, it only attempts to schedule it to run on the device that will maximize both energy efficiency and user experience.
2. The Cloud Receiver receives a task from the cloud, it assesses that it can execute the workload (via the Availability Checker) and accepts or declines accordingly. If more than one device accepts a workload, the availability will be the tie-breaker (FT and FMT will always have highest availability).

Giving Android the HYCCUPS

The main reason which convinced us to choose the Android platform is its openness. Not only does it provide stable and easy to use APIs and tools, but it also provides the source code for the internal implementation of the framework itself, thus offering insights for us in implementing and integrating HYCCUPS in Android.

Before we can go any further, we feel obliged to present some basic notions for developing applications in Android. The Android software stack is constructed based on the following building blocks:

- **Activity**: represents a user interaction component. It's usually defined as a focused action that a user may perform. An Activity is responsible for creating a window where user interaction elements are inserted and which supports interaction events.
- **Service**: represents a background worker. Its main responsibilities include: performing of longer computation operations without user interaction, or supplying functionality for other components or even other applications.
- **Content Provider**: manages access to structured sets of data, usually SQLite databases. Similarly to Services, Content Providers may be spanned over multiple applications.
- **Broadcast Receiver**: base component used for asynchronous events which are actually manifested by means of Intents. Broadcast Receivers are registered into the Android system on behalf of applications, and are used to filter and react only when receiving certain types of Intents.
- **Intent**: represents the glue between Android components. Represented as structured messages, Intents are used to implement an inter-component communication (ICC), a process much similar to UNIX inter-process communication (IPC).

Considering that HYCCUPS will have to extend the Android API in order to offer a transparent middleware layer, we have taken much interest in the Intent ICC process. As Intents are mostly used to broadcast messages throughout the system as to asynchronously notify components of actions that need to be offloaded towards them, it seems to be the most likely candidate for our extensions. In order for our extensions to be clearly understood, we are obliged to provide a description of Intents and their behavior.

Basically, an Intent is a structured message which may contain the following data:

- **action**: the general action to be performed;
- **data**: an URI expressing the general data on which the action will be performed;
- **category**: gives additional data about the action to be performed;
- **type**: specifies an explicit MIME type for the data;
- **component**: explicit name of a component which will treat this intent;
- **extras**: supplementary data attached to an intent.

For the ICC process to be complete, the system must be able to identify the receiver for a specific intent, process denominated as **Intent Resolution**. According to Meier (2010), intent resolution is comprised of the following steps:

1. the system aggregates all intent filters that were registered;
2. intent filters which do not match the action or category are dropped;
    1. actions match if both are alike or no action was provided;
    2. all categories defined in an intent must have a correspondent in the filter;
3. the intent data URI is compared to the filter's data scheme; any difference in scheme, host/authority, MIME type, automatically removes the intent filter;
4. if more components are resolved to an intent, a list of comprising candidates is offered in return.

As can be seen, the Intent ICC design principle encourages decoupling and extensibility of components and applications. Furthermore, it proves to be the component which HYCCUPS needs to extend, in order for a task to cross the bounds of the device itself and into the cloud. A significant issue regarding the Android extension is formulated now as to reflect our design considerations: in order for HYCCUPS to be able to run on (mostly) all versions of Android, we cannot modify the software stack itself. In consequence, we use the Intent ICC mechanism to cover the internals of HYCCUPS and to offer a transparent middleware layer for developers of applications.

The following components represent the Android extension of Intents and Broadcast Receivers used to support HYCCUPS:

1. **Cloud Intent**: an Intent container which adds an extra action and extra component (the extra fields in the Cloud Intent are used instead of the original ones). The main action of the Intent is CLOUD INTENT and the extra action is used in the Intent Resolution mechanism in the cloud to substitute for the main one. The main component of the Intent will be set to Cloud Receiver, and the extra component is provided as to substitute for the main one used by HYCCUPS. At this point, we separate Cloud Intents into two categories: Selfish Goal Intents and Common Goal Intents. This separation stems from the duality of HYCCUPS: it can be either used for tapping into distributed mobile data disregarding energy consumption, or it can be used to achieve energy efficiency by collaboration.
2. **Cloud Receiver**: a Broadcast Receiver registered into the system and waiting for CLOUD INTENT actions. This component is responsible for reacting to cloud intents, internal (from the device) or external (from the cloud). This component will further interact with the HYCCUPS framework to offload execution into the cloud.

In order for the mobile cloud to react seamlessly and coherently, an efficient Android peer-to-peer communication solution is needed to fulfill all responsibilities of communication in the cloud. In this respect, we have selected AllJoyn as our communication framework.

AllJoyn is an open source software framework which offers a peer-to-peer communication environment for heterogeneous distributed systems across different device classes with emphasis on mobility, security and dynamism, by implementing the D-Bus protocol. It provides an abstraction layer allowing it to run on multiple operating systems such as Linux distributions (Ubuntu included) and most versions of Microsoft Windows (Windows 7 included). Such a feature will allow us to register wearable computers into the mobile cloud as Fixed Terminals, in order to provide powerful offloading of mobile execution onto them as illustrated in Figure 3.

AllJoyn's main goal is to provide a software bus that offers distributed advertising and discovery services in a secure mobile environment. In addition, the system supplies a Java-like location-transparent RMI. In the AllJoyn world, the most significant abstraction is the software bus, which connects and glues together all the clients and services, and implicitly all of the wearable devices. Being a peer-to-peer system, all peers connect to the aforementioned bus by means of bus attachments. The bus attachment is another software abstraction that exists in the peer space and actually represents the link to the inter-process communication that connects clients and services alike to the AllJoyn daemon. At the time of connection, each bus attachment is assigned a unique name by the system and, if advertising is needed, the aforesaid bus attachment may request a unique well-known name, by means of which it can make itself public to the rest of the distributed system. Well-known names are human-readable names which reside in a reversed-domain, like a self-managed namespace. The sheer existence of a bus attachment of a specific name directly implies the existence of at least one instance of a bus object implementing the interface specified by said name. Bus objects are organized in a hierarchy in which the bus attachment is root, similar to a UNIX file system. An AllJoyn interface consists of bus methods, bus properties and bus signals. Bus methods are traditional RMI method implementations, properties are actual fields accessed by remote setters and getters, and signals are similar to methods, except that they do not expect any results (they are more akin to broadcasting). Services implement interfaces through bus objects and advertise their possibilities by means of well-known names. Clients are expected to use proxy bus objects for ease of interaction with the service, by means of simple interfaces.

In order for AllJoyn to work on Android, it is sufficient to install the AllJoyn Daemon: the glue that holds the entire distributed system together and which runs in the background, without user focus, waiting for events and reacting to them accordingly. The Daemon's implementation provides an operating system abstraction layer, which represents a wrapper over the native binaries, and upon which the entire transport system is built. The OS abstraction layer actually provides the necessary abstractions for implementing the platform-neutral feature, so that the daemon may be run on Linux, Windows and Android alike. Moreover, the security issues of HYCCUPS are covered by AllJoyn through using authentication with x509 certificates.

EXPERIMENTAL RESULTS

Evaluating such a complex system as HYCCUPS proved to be more difficult than anticipated. The issue did not arise due to the arduousness of implementing and deploying the actual framework for a specific smartphone platform (as the previous Section showed that a port to Android is straightforward), but because of the lack of applications implemented for our framework. In order to thoroughly test it, we need to create a vast number of Android applications that take advantage of the HYCCUPS cloud and which pose interest to the users.

As was noticed in the tracing experiment in Marin at al. (2012), the greatest pitfall of such endeavors is the lack of conscientiousness of participants. In order to attempt such an experiment, both volunteers and application developers alike need to be convinced of the benefits that HYCCUPS offers. Without the support of both aforementioned parties, such an endeavor will be in vain.

To overcome said issues, we have developed the HYCCUPS Emulator, which provides valuable insight into the inner-workings and performances of the entire framework. It makes use of the traced contextual data described in the tracing experiment from Marin et al. (2012), which was proven to be sufficient for our needs, by means of the methodology and guidelines provided in Marin et al. (2013).

The HYCCUPS Emulator is implemented in Java, as to ease the porting of the algorithm from Android (which also uses Java, but also proprietary APIs). The traced data was stored in a MySQL database, and the connection to the emulator was made through the *Connector/J Java Database Connectivity* (JDBC) driver. The emulator also makes use of the *Efficient Java Matrix Library* (EJML) for implementing linear regression (by using the normal function as previously explained).

The emulator is built based on the following assumptions:

1. All devices use the same reference for time: due to the fact that the devices involved in the tracing dataset did not use a network clock, we must assume that they all are synchronized to the same time reference.
2. All environmental events are synchronous and sequentially executed for all users. As such, each event for any device acts as a barrier for all users. By so doing, we are able to implement deterministic heuristics for generating repeatable workloads.
3. All devices have only a single core: the tracing data did not include anything about the number of available cores or about the number of online cores.

4. There is no network transfer delay: all of the workloads generated are purely computational. The emulator does not assume that workloads are sending data as well, instead it presumes that the data is already present on the device. Therefore, all workloads are expressed in CPU cycles.
5. The bandwidth of the network is considered to be unlimited.

Workloads are generated in a deterministic manner so as to be repetitive, using a heuristic that takes into account the following features: the user of the device is active, or the device is connected to the AllJoyn bus. The main difficulty with working with the tracing dataset is the lack of conscientiousness of the volunteers. Because of this, there are not many nodes actively involved in offloading, as only three devices out of all 65 participants seem to be collaborating in an energetic manner. This will impact the results, as will be seen in the remainder of this Section.

We have attempted to cover multiple types of workloads while analyzing the HYCCUPS framework. We have generated three scenarios using a total of 726 tasks over 2 months of emulator execution, which are sent in a pseudo-random fashion to all active users in the system, with the following computational requirements:

1. 100Mcycle tasks: small tasks which are easily executed by any node. Such tasks can be associated to computing a move in a game of chess.
2. 1Gcycle tasks: regular-sized tasks which should be executed by most devices. This scenario could be associated with computing the value of π.
3. 10Gcycle tasks: large tasks which should be a burden for all devices. These tasks could relate to the analysis of satellite images (given that the images are already downloaded).

Figures 4, 5, and 6 show the total debt accumulated by all users on a daily basis for all three scenarios. As can be seen in all three cases, the cost model gives better performance, as it seems that altruism serves the community better than leeching off of other nodes. Furthermore, Table 2 shows that, even though working without a cost model can actually offload more, the overall benefit of using such a model is much greater, as it actually saves more computing time. As can be seen, HYCCUPS manages to save valuable computing, which is actually proportional to the size of the task. The reader should be advised that, although at a first glance the amount of saved time does not seem to be large, only three devices were actively offloading tasks. This should generally change the perspective over the results. Also, since more than a quarter of the tasks were executed remotely on devices that were charging, the actual computing power save increases. Moreover, out of the 57(55) offloaded tasks, none of them failed to complete, which proves that the adaptive and predictive models used in collaboration with the contextual search algorithm are working properly and correctly.

Last, but not least, charts in Figures 7, 8, and 9, illustrate the total execution time as tasks are sequentially run. Moreover, they compare the HYCCUPS execution model with the current traditional Android model. As can be observed, the HYCCUPS framework speeds up mobile applications by means of offloading onto remote idle nodes. However, these figures do not contain the execution for all of the 726 tasks, but only for the 57(55) offloaded tasks. We are not interested in the tasks that are always run locally, as HYCCUPS can only optimize workloads that actually have the possibility to be offloaded (although the predictors might have missed some offloading opportunities).

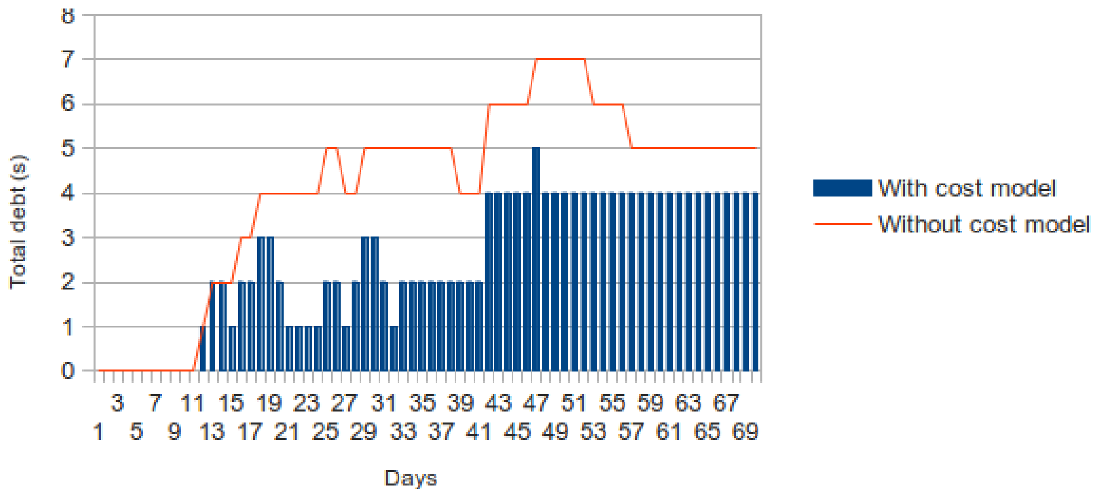

Figure 4: Total accumulated debt, for 100 MCycle tasks.

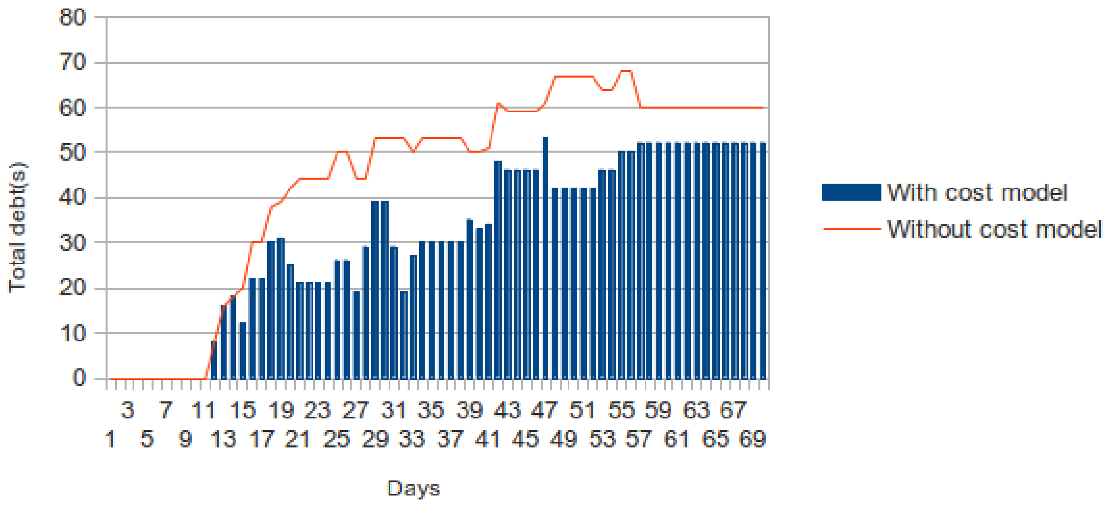

Figure 5: Total accumulated debt, for 1Gcycle tasks.

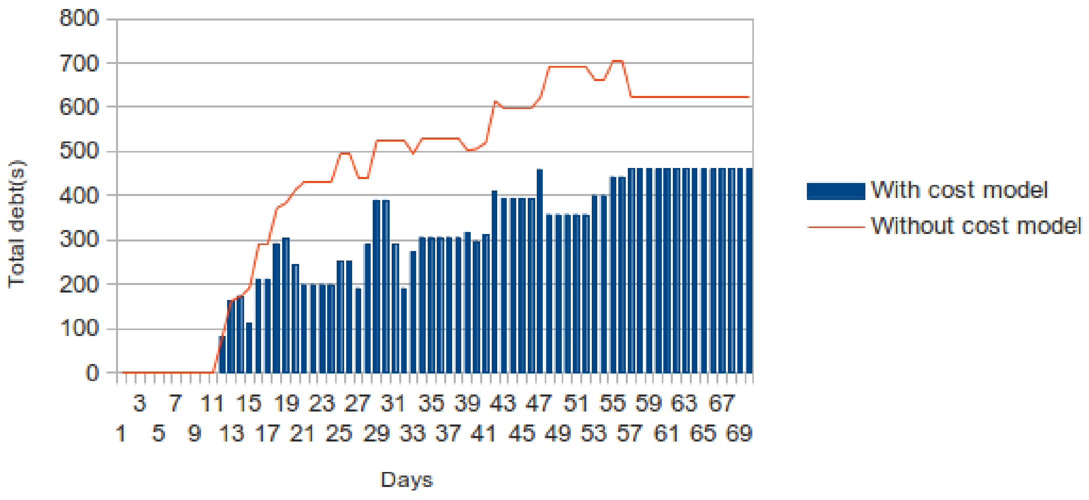

Figure 6: Total accumulated debt, for 10 GCycle tasks.

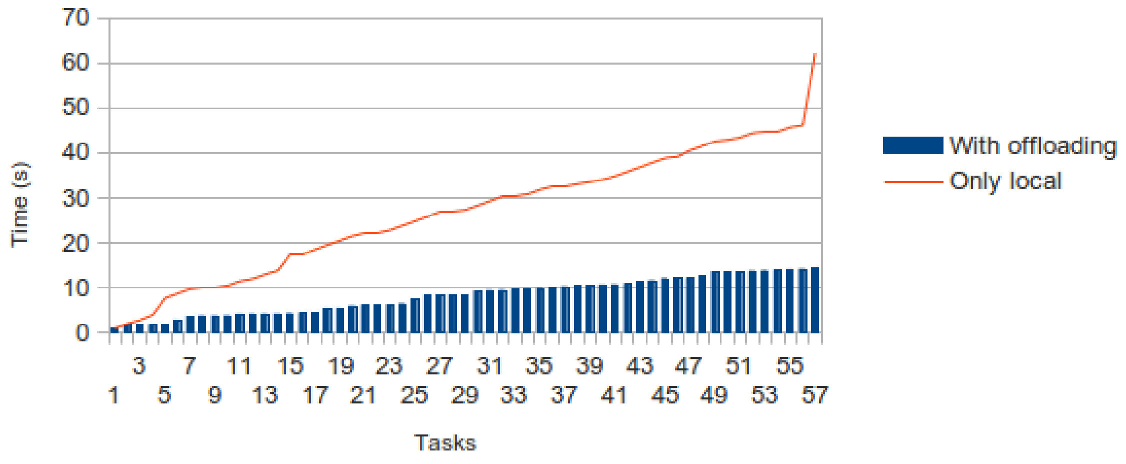

Figure 7: Total accumulated execution time for offloaded tasks, for 100 MCycle tasks.

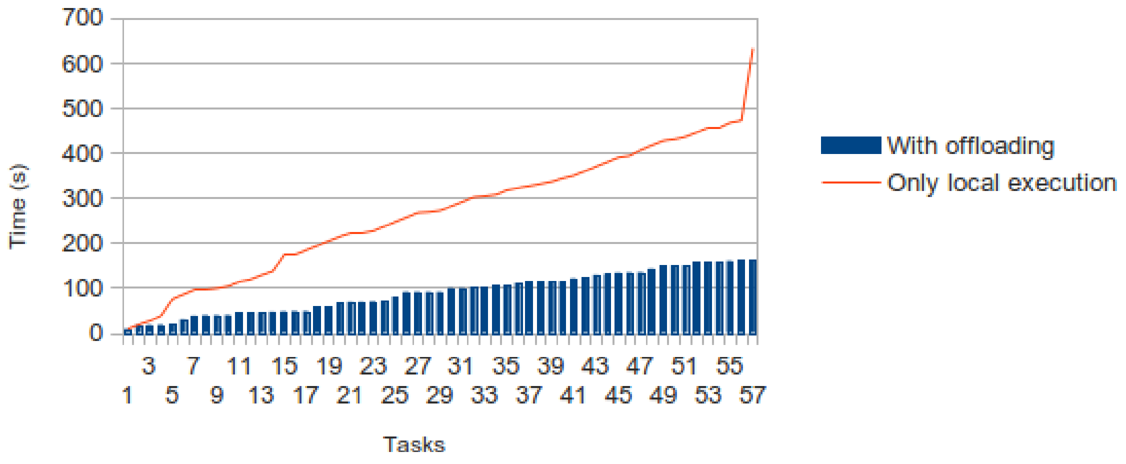

Figure 8: Total accumulated execution time for offloaded tasks, for 1 GCycle tasks.

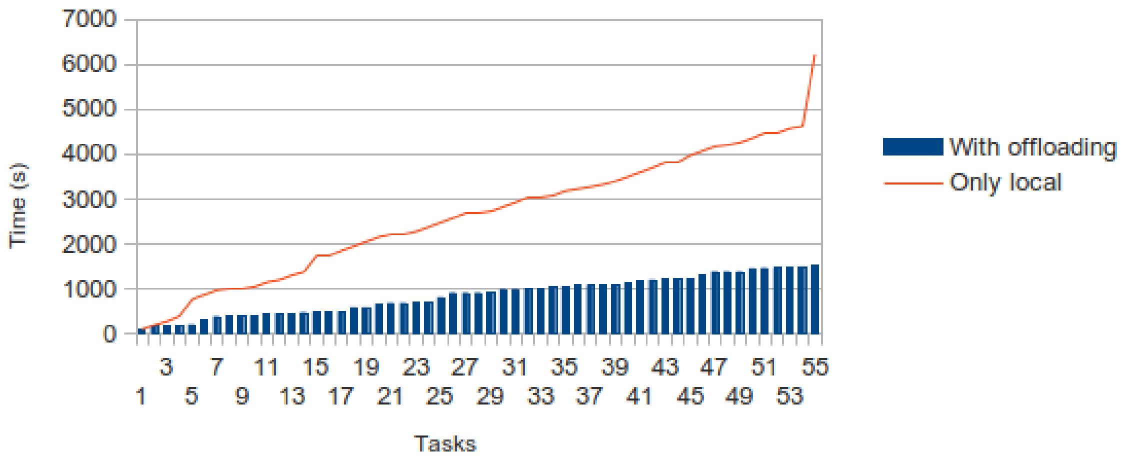

Figure 9: Total accumulated execution time for offloaded tasks, for 10 GCycle tasks.

Scenario	Cost model?	Total offloaded	Failed offloads	FMT offloads	Total saved time (s)
100 Mcycle	Y	57	0	16	48000
100 Mcycle	N	57	0	16	45000
1 Gcycle	Y	57	0	15	472000
1 Gcycle	N	57	0	16	448000
10 Gcycle	Y	55	0	15	4723000
10 Gcycle	N	57	0	16	4481000

Table 2:Offloading statistics for all three scenarios

Although the above results are promising, the simulation process is limited, as it does not take network resources into consideration. Opportunistic networking is based on intermittent connectivity and lack of a well-established communication infrastructure and, as such, network resources cannot be ignored in assessing the performance of opportunistic offloading models. Unfortunately, the tracing experiment in Marin et al. (2012) does not provide any networking statistics, which is the main reason behind assumptions (4) and (5) that the HYCCUPS Emulator is based on. To overcome this issue, we introduce a synthetic model which randomly generates and injects network throughput events into the simulation process. The model is based on the *802.11g* protocol, by considering a 2.4 GHz bandwidth and throughput of up to 4000 KB/s.

The presence of the networking throughput in the design process directly impacts the *mobile-to-mobile contextual offloading* model, as the definition of the computing potential is revisited: the time required to transfer the input data needed by the task in current and predicted networking conditions, along with the time required by the processor to run a workload considering current and predicted values for CPU usage. As such, an offloaded task is remodeled not only to describe its computational needs (e.g. CPU cycles), but also the network resource usage incurred by the dimension of its transferred input data. To better understand both the impact of network transfers on offloading, and also of the relationship between task input size and offloading success rate, we have performed parameter sweeps on the simulation of the HYCCUPS framework, by varying the input size and network throughput, respectively the input data size and the computational needs of tasks.

The contextual offloading model is heavily impacted by network transfers, as Figures 10 and 11 show that the number of successfully offloaded tasks is halved (from 40 down to 20 offloads) when the input size reaches its maximum value. As expected, the best case scenario is when the tasks are heavily computational and do not require any network transfers. On the other hand, as the input data size increases linearly, the number of offloaded tasks decreases at quadratic rates. Interestingly enough, the number of offloaded tasks levels out when the task's computational needs are higher than 40 Mcycles and input data size is higher than 4MB. This proves that the contextual offloading model is still able to provide undeniable energy efficiency in situations closer to real-life. Moreover, Figure 10 illustrates the influence of network transfers on the total time saved through contextual offloading; as expected, the saved time increases linearly, proportional to computational needs. However, when increasing the input data size of tasks, the overall speedup of applications decreases linearly, but with a fairly smaller slope. This is further evidence of the increased performance of the offloading model, which, in the case of highly computational tasks that require large transfers, it is able to provide impressive application speedup, by saving up to about 400 min out of the almost 700 min saved in the base case that involves no network transfers.

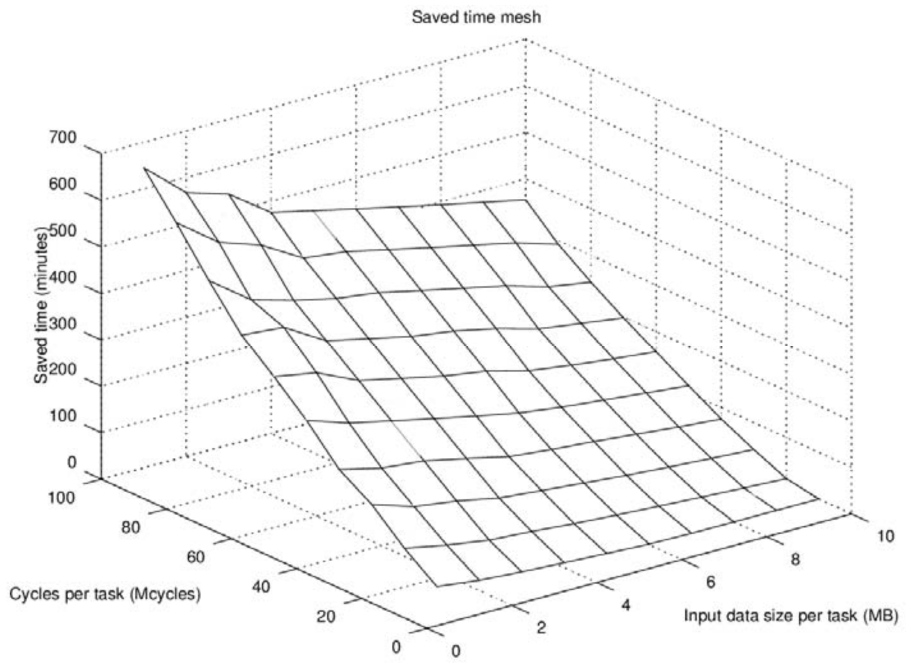

Figure 10: Total saved time, for varying CPU cycles and input data size.

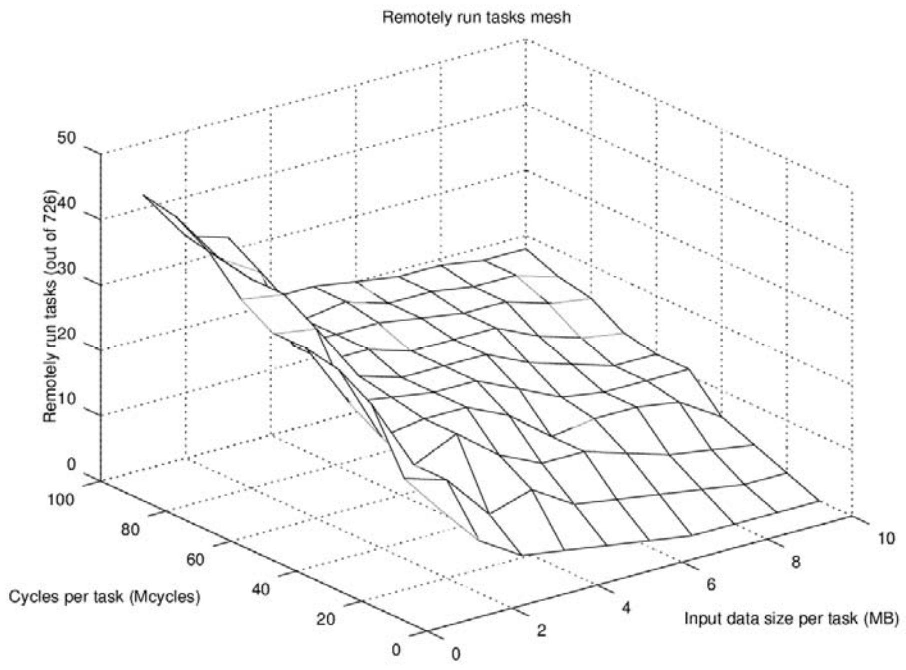

Figure 11: Remote run tasks for varying CPU cycles and input data size.

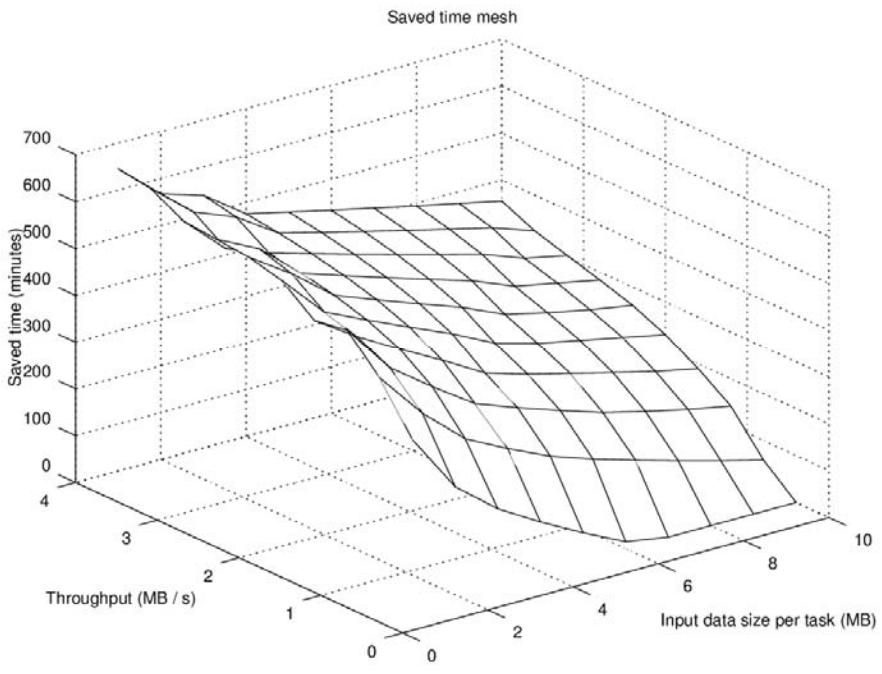

Figure 12: Total saved time for varying throughput and input data size.

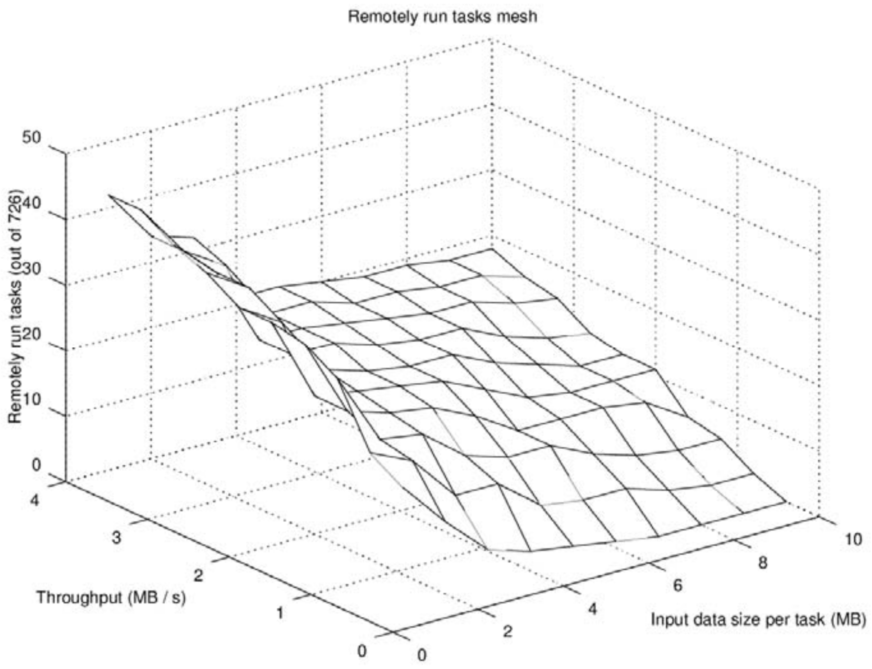

Figure 13: Remote run tasks for varying throughput and input data size.

As previously stated, the actual communication infrastructure is highly important in opportunistic networking. By also varying the throughput of the underlying communication channels, Figure 13 illustrates how the offloading model is able to handle real-world conditions. As also observed in Figures 10 and 11, the number of successfully offloaded tasks sensibly decreases when the wireless networks are experiencing increasing traffic, but it still manages to level out when the throughput is higher than 2 MB/s and input data size is higher than 4MB. As opposed to task size, the network load has less influence when the throughput has acceptable rates (higher than 2MB/s), as it decreases with a more acceptable downward slope. However, when the data input size of tasks is very high, the actual time saved through offloading experiences a harsh drop, this being the worst-case scenario for offloading, as all tasks will end up being run locally.

FUTURE WORK

Throughout all of the stages of designing, implementing and evaluating our solution, one of the foremost challenges has been dealing with the incomplete tracing data sets. As previously mentioned, the lack of conscientiousness of participants in the tracing experiments we have conducted influenced our entire research, by obliging us to make use of techniques that deal with uncertainty. We believe that more complete traces will prove to be of invaluable importance for both developing real-life models which better describe realistic situations that mobile users are faced with, as well as for evaluating solutions in opportunistic networking and computing. Therefore, we plan on conducting future tracing experiments aimed at raising the interest and awareness about the benefits of collecting contextual data. We believe that attracting and encouraging mobile users towards participating in such actions will lead to more complete tracing data sets, which, in turn, will generate the opportunity for creating novel context-aware applications.

Although the HYCCUPS Emulator has proved to be an invaluable tool in determining the performance of our solution, it is fairly limited in its functionality, as it only resumes to playing back real-world traces. Based on predictive models extracted from the contextual data collected from mobile users, we envision a distributed simulation platform which can simulate both opportunistic networks and real mobile device usage. This platform will use real mobile emulators running HYCCUPS, which are connected through a virtual environment modeling wireless access points and real time opportunistic interactions. Because such interactions are out of the control of the user or device, systems like HYCCUPS must rely on the natural movement of large numbers of individuals. This makes testing opportunistic systems in the real world not feasible, so simulators have to be used. Although some opportunistic network simulators have been developed, in order to validate HYCCUPS, there are a number of requirements which are not currently met by any of them. First of all, the opportunistic network modeling is clearly fundamental, since it has to simulate the environment for the entire platform based on real-world opportunistic traces and models. Second, the simulator must support a number of other types of usability events and monitor statistics such as CPU load and battery levels. Finally, the simulator must provide an environment as close to the real world as possible and, in order to do this, actual mobile emulators have to be used, which perform tasks as scheduled by the HYCCUPS framework in real time. This is as close as we can get to a real world scenario without using physical devices.

Last, but not least, successfully deploying our solution to the public is the most important direction on which we are focusing. Such an endeavor will probably prove to be the most challenging, as the HYCCUPS framework by itself is useless as long as there are no popular applications making use of it. Therefore, for a guaranteed success, creating applications based on our framework that pose interest to both users and developers is considered of paramount importance.

CONCLUSIONS

We believe that current models in the design, development and deployment of mobile applications are becoming inadequate, as they do not take sustainability into account. Furthermore, we consider the endeavors currently taken in mobile cloud computing to be incomplete and limited. As such, the need for novel computational models that focus on power saving is growing exponentially. We attempt to fill this requirement by means of the HYCCUPS framework.

HYCCUPS introduces intelligent collaboration, a new computational model which offers feasible synergy between interacting peers in opportunistic networks by correlating mobility with availability information by means of a contextual search algorithm. Moreover, the algorithm is adaptive to environmental changes so as to improve battery health, maximize power save, minimize overall execution time of mobile applications and, last, but not least, to preserve or even enhance user experience.

We design a simple deployment model hidden behind language constructs specific to the smartphone platform on top of which it is distributed. By so doing, we encourage researchers and application developers to design and implement applications to make use of HYCCUPS. Moreover, such a deployment model does not require any additional training for the developers, as they will be using the same APIs in creating applications. We also enforce modularity into the development process, as programmers will have to implement both the GUI components, as well as the task executors. This component symmetry will also make the code easier to understand, as it is presented in the current traditional method of designing applications.

In conclusion, we believe that the HYCCUPS framework will render better results, considering both sustainability and efficiency, as it is based upon a novel intelligent *mobile-to-mobile contextual offloading* model, and will offer on-the-fly cloud capabilities to sets of interconnected smartphones. Moreover, it will allow seamless development of Android applications that can take advantage of such a contextual community.

ACKNOWLEDGEMENT

The research presented in this paper is supported by national project MobiWay, PN-II-PT-PCCA-2013-4-0321. The work is aligned with COST Action IC1303, “Algorithms, Architectures and Platforms for Enhanced Living Environments (AAPELE)”, and COST Action IC1406, “High-Performance Modelling and Simulation for Big Data Applications (cHiPSet)”.

REFERENCES

Agarwal, B., Chitnis, P., Dey, A., Jain, K., Navda, V., Padmanabhan, V. N., Ramjee, R., Schulman, A. & Spring, N. (2010). Stratus: energy-efficient mobile communication using cloud support.. In S. Kalyanaraman, V. N. Padmanabhan, K. K. Ramakrishnan, R. Shorey & G. M. Voelker (eds.), *SIGCOMM* (p./pp. 477-478), : ACM. ISBN: 978-1-4503-0201-2

Anand, B., Thirugnanam, K., Sebastian, J., Kannan, P.G., Ananda, A.L., Chan, M.C.,, & Balan, R.K. (2010), Adaptive display power management for mobile games. In Proceedings of the 9th international conference on Mobile systems, applications, and services, MobiSys '11, pages 57-70, New York, NY, USA

Balasubramanian, N., Balasubramanian, A. & Venkataramani, A. (2009). Energy consumption in mobile phones: a measurement study and implications for network applications.. In A. Feldmann & L. Mathy (eds.), *Internet Measurement Conference* (p./pp. 280-293), : ACM. ISBN: 978-1-60558-771-4

Barabasi, A.-L. (2005). The origin of bursts and heavy tails in human dynamics. *Nature*, 435, 207

Bisong, A., & Rahman, S. M. (2011). An Overview of the Security Concerns in Enterprise Cloud Computing. *CoRR*, abs/1101.5613

Challam, V., Gauch, S. & Chandramouli, A. (2007). Contextual Search Using Ontology-Based User Profiles.. In D. Evans, S. Furui & C. Soulé-Dupuy (eds.), *RIAO*, : CID

Chun, B.-G., & Maniatis, P. (2010), Dynamically partitioning applications between weak devices and clouds. In Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond, MCS '10, pages 7:1-7:5, NewYork, NY, USA, ACM

Chun, B.-G., Ihm, S., Maniatis, P., Naik, M. & Patti, A. (2011). CloneCloud: elastic execution between mobile device and cloud.. In C. M. Kirsch & G. Heiser (eds.), *EuroSys* (p./pp. 301-314), : ACM. ISBN: 978-1-4503-0634-8

Cuervo, E., Balasubramanian, A., ki Cho, D., Wolman, A., Saroiu, S., Chandra, R. & Bahl, P. (2010). MAUI: making smartphones last longer with code offload.. In S. Banerjee, S. Keshav & A. Wolman (eds.), *MobiSys* (p./pp. 49-62), : ACM. ISBN: 978-1-60558-985-5

Hoang, D. T., Lee, C., Niyato, D. & Wang, P. (2013). A survey of mobile cloud computing: architecture, applications, and approaches.. *Wireless Communications and Mobile Computing*, 13, 1587-1611

Holly, R. (2011), *How much of your phone is yours?*, Retrieved May 2014 from http://www.geek.com/mobile/how-much-of-your-phone-is-yours-1440611/

Hong, J., Suh, E. & Kim, S.-J. (2009). Context-aware systems: A literature review and classification.. *Expert Syst. Appl.*, 36, 8509-8522

Huerta-Canepa, G., & Lee, D. (2010), A virtual cloud computing provider for mobile devices. In *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond* (MCS '10). ACM, New York, NY, USA, , Article 6 , 5 pages. DOI=10.1145/1810931.1810937 http://doi.acm.org/10.1145/1810931.1810937

Kemp, R., Palmer, N., Kielmann, T., and Bal, H. (2012). Cuckoo: A computation offloading framework for smartphones mobile computing, applications, and services. volume 76 of Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, chapter 4, pages 59–79. Springer Berlin Heidelberg, Berlin, Heidelberg

Kraft, R., Maghoul, F. & Chang, C.-C. (2005). Y!Q: contextual search at the point of inspiration.. In O. Herzog, H.-J. Schek, N. Fuhr, A. Chowdhury & W. Teiken (eds.), *CIKM* (p./pp. 816-823), : ACM. ISBN: 1-59593-140-6

Liang, W.-Y., & Lai., P.-T. (2010), Design and implementation of a critical speed-based DVFS mechanism for the Android operating system. In Embedded and Multimedia Computing (EMC), 2010 5th International Conference on, pages 1-6, Cebu, Philippines

Liang, W.-Y., Lai, P.-T., & Chiou, C. (2010), An energy conservation DVFS algorithm for the Android operating system. Journal of Convergence, 1(1):93-100

Marin, R.-C. & Dobre, C. (2013), Reaching for the clouds: contextually enhancing smartphones for energy efficiency., *in* Constandinos X. Mavromoustakis; Tasos Dagiuklas & Lei Shu, ed., 'HP-MOSys' , ACM, , pp. 31-38

Marin, R.-C., Dobre, C. & Xhafa, F. (2012). Exploring Predictability in Mobile Interaction.. *EIDWT* (p./pp. 133-139)

Marin, R.-C., Dobre, C. & Xhafa, F. (2014), A methodology for assessing the predictable behaviour of mobile users in wireless networks. Concurrency Computat.: Pract. Exper., 26: 1215–1230. doi: 10.1002/cpe.3064

Marinelli, E.E. (2009), *Hyrax: Cloud Computing on Mobile Devices using MapReduce*, Unpublished Masters dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh, USA

Meier, R. (2010). *Professional Android 2 Application Development*. 1st ed.

Minkov, E., Cohen, W. W. & Ng, A. Y. (2006). Contextual search and name disambiguation in email using graphs.. In E. N. Efthimiadis, S. T. Dumais, D. Hawking & K. Järvelin (eds.), *SIGIR* (p./pp. 27-34), : ACM. ISBN: 1-59593-369-7

Murray, D. G., Yoneki, E., Crowcroft, J. & Hand, S. (2010). The case for crowd computing.. In L. P. Cox & A. Wolman (eds.), *MobiHeld* (p./pp. 39-44), : ACM. ISBN: 978-1-4503-0197-8

Peralta, A. (2011), *Samsung, HTC And Carrier IQ Face Suit Over Logging Software*, Retrieved December 2011 from http://www.npr.org/blogs/thetwo-way/2011/12/02/143051586/samsung-htc-and-carrier-iq-face-suit-over-logging-software

Robison, W. J. 2010. Free at what cost?: cloud computing privacy under the Stored Communications Act. *Georgetown Law Journal*, 98(4). http://papers.ssrn.com/sol3/Delivery.cfm/SSRN_ID1596975_code1461162.pdf?abstractid=1596975&mirid=2

Schlachter, F. (2012), Has the Battery Bubble Burst?, *APSNews*, 8(21), 8. Retrieved October 2012 from http://www.aps.org/publications/apsnews/

Schulman, A., Navda, V., Ramjee, R., Spring, N., Deshpande, P., Grunewald, C., Jain, K. & Padmanabhan, V. N. (2010). Bartendr: a practical approach to energy-aware cellular data scheduling.. In N. H. Vaidya, S. Banerjee & D. Katabi (eds.), *MOBICOM* (p./pp. 85-96), : ACM. ISBN: 978-1-4503-0181-7

Song, C., Qu, Z., Blumm, N. & Barabási, A.-L. (2010). Limits of Predictability in Human Mobility. *Science*, 327, 1018--1021. doi: 10.1126/science.1177170

Weiser, M. (1991), The Computer for the 21st Century, *Scientific American* 265 (3) , 66-75

Zhang, X., Kunjithapatham, A., Jeong, S. & Gibbs, S. (2011). Towards an Elastic Application Model for Augmenting the Computing Capabilities of Mobile Devices with Cloud Computing.. *MONET*, 16, 270-284

Zhang, X., Schiffman, J., Gibbs, S., Kunjithapatham, A. & Jeong, S. (2009). Securing elastic applications on mobile devices for cloud computing.. In R. Sion & D. Song (eds.), *CCSW* (p./pp. 127-134), : ACM. ISBN: 978-1-60558-784-4

Key Terms and Definitions:

Context-awareness is a property of mobile devices that is defined complementarily to location awareness. Whereas location may determine how certain processes in a device operate, context may be applied more flexibly with mobile users, especially with users of smart phones. Context awareness originated as a term from ubiquitous computing or as so-called pervasive computing which sought to deal with linking changes in the environment with computer systems, which are otherwise static.

Mobile Cloud Computing (MCC) is the combination of cloud computing, mobile computing and wireless networks to bring rich computational resources to mobile users, network operators, as well as cloud computing providers. The ultimate goal of MCC is to enable execution of rich mobile applications on a plethora of mobile devices, with a rich user experience. MCC provides business opportunities for mobile network operators as well as cloud providers.

Energy efficiency is the goal to reduce the amount of energy required to provide products and services. There are many motivations to improve energy efficiency. Reducing energy use reduces energy costs and may result in a financial cost saving to consumers if the energy savings offset any additional costs of implementing an energy efficient technology. Reducing energy use is also seen as a solution to the problem of reducing carbon dioxide emissions.

Mobile-to-mobile contextual offloading is a concept in which handhelds make use of a contextual search algorithm to schedule the remote execution of tasks in trusted smartphone communities based on predicting the availability and mobility of nearby devices

A **Cloud Intent** is an Intent container which adds an extra action and extra component (the extra fields in the Cloud Intent are used instead of the original ones). The extra action is used in the Intent Resolution mechanism in the cloud to substitute for the main one.

A **Cloud Receiver** is a Broadcast Receiver registered into the system and waiting for Cloud Intent actions. This component is responsible for reacting to cloud intents, internal (from the device) or external (from the cloud). This component further interacts with the HYCCUPS framework to offload execution into the cloud.

AllJoyn is an open source software framework which offers a peer-to-peer communication environment for heterogeneous distributed systems across different device classes with emphasis on mobility, security and dynamism, by implementing the D-Bus protocol. It provides an abstraction layer allowing it to run on multiple operating systems such as Linux distributions (Ubuntu included) and most versions of Microsoft Windows (Windows 7 included).

A **Mobile Terminal** (MT) is a smartphone terminal, not plugged in. Its main characteristic is mobility. Not always available for executing workloads.

A **Fixed Mobile Terminal** (FMT) is a plugged-in smartphone terminal. Acts as a Fixed Terminal, but may rapidly return to a mobility state (MT).

A **Fixed Terminal** (FT) is a wearable computer, and will always be available for executing workloads. This is an optional component, as it raises compatibility issues: the developer has to explicitly redesign the mobile application code so it can be deployed on other platforms.

Opportunistic mobile networks consist of human-carried mobile devices that communicate with each other in a store-carry-and-forward fashion, without any infrastructure. Compared to the classical networks, they present distinct challenges. In opportunistic networks, disconnections and highly variable delays caused by human mobility are the norm rather than an exception. The solution consists of dynamically building routes, as each node acts according to the store-carry-and-forward paradigm. Thus, contacts between nodes are viewed as opportunities to move data closer to the destination. Such networks are therefore formed between nodes spread across the environment, without any knowledge of a network topology. The routes between nodes are dynamically created, and nodes can be opportunistically used as a next hop for bringing each message closer to the destination. Nodes may store a message, carry it around, and forward it when they encounter the destination or a node that is more likely to reach the destination.