UNIUNEA EUROPEANĂ

GUVERNUL ROMÂNIEI
MINISTERUL MUNCII, FAMILIEI,
PROTECȚIEI SOCIALE ȘI
PERSOANELOR VÂRSTNICE
AMPOSDRU

Fondul Social European
POSDRU 2007-2013

Instrumente Structurale
2007-2013

MINISTERUL
EDUCAȚIEI
NAȚIONALE
OIPOSDRU

Universitatea
POLITEHNICA
din București

FONDUL SOCIAL EUROPEAN
Investeşte în oameni!
Programul Operaţional Sectorial pentru Dezvoltarea Resurselor Umane 2007 – 2013
Proiect POSDRU/107/1.5/S/76909 – *Valorificarea capitalului uman din cercetare prin burse doctorale (ValueDoc)*

UNIVERSITATEA **POLITEHNICA** DIN BUCUREŞTI

Facultatea de Automatică și Calculatoare

Departamentul Calculatoare

Nr. Decizie Senat 225 din 27.09.2013

TEZĂ DE DOCTORAT

Extinderea Capabilităților Dispozitivelor Mobile
prin Transferarea Sarcinilor în Cloud

Extending the Capabilities of Mobile Devices
through Cloud Offloading

Autor: As.ing. Olteanu Alexandru-Corneliu

Conducãtor de doctorat: Prof.dr.ing. Nicolae Țăpuș

COMISIA DE DOCTORAT

Președinte	Prof.dr.ing. Adina Florea	de la	Universitatea Politehnica București
Conducător de doctorat	Prof.dr.ing. Nicolae Țăpuș	de la	Universitatea Politehnica București
Referent	Prof.dr.ing. Victor Patriciu	de la	Academia Tehnică Militară București
Referent	Prof.dr.ing. Ion Ivan	de la	Academia de Științe Economice București
Referent	Prof.dr.ing. Valentin Cristea	de la	Universitatea Politehnica București

Bucureşti, 2013

This work is funded by the Sectoral Operational Programme
Human Resources Development 2007-2013 of the Romanian Ministry
of Labour, Family, and Social Protection through
the Financial Agreement POSDRU1071.5S76909.

This thesis is the result of a three year journey, with more than eight months of internships, in the Netherlands and Germany. I found, both at home and abroad, many new colleagues and friends. I would like to thank them all for their influence on the outcome of my PhD work, and ultimately on this thesis. I will mention some names briefly, as space dictates, and, if your name has been left out, please accept my non-nominative thanks.

I would like to start by expressing my gratitude to Nicolae Țăpuș, Alexandru Iosup and Sven Zeisberg, my advisors throughout the three years of the PhD and to thank my colleagues and collaborators from University Politehnica of Bucharest, from Delft University of Technology and from the University of Applied Sciences in Dresden.

Any graduate of the Computer Science and Engineering Department in University Politehnica of Bucharest can surely say that he or she had at least an activity under the guidance of Prof. Nicolae Țăpuș. I would like to thank him for accepting me as a PhD student, which I consider a mark of our mutual consideration and I will always hold with a high regard. As a Teaching Assistant, I came in contact with many fellow researchers and graduate students. I would like to mention my co-authors—Dan Tudose, Dan Rizea, Alex Gherghina, Alex Făsui, Virgil Zamfirache, Răzvan Prejbeanu and Șerban Chiricescu—for our mutual support and our continuous collaboration. I also want to thank my colleagues and friends—Adriana Drăchici, Andrei Voinescu, Laura Gheorghe, Dan Dragomir, Silvia Stegaru, Răzvan Dobre, Alex Herișanu, Emil Slușanschi and Răzvan Rughiniș—for providing a stimulating research environment.

Advancing in chronological order, my first internship was at the University of Applied Sciences in Dresden. My special thanks go to Sven Zeisberg for this opportunity and for personally getting involved to make everything work. I would also like to thank my main collaborators—Frank Stephan and Alfred Reisner— as well as the guys involved with the Communications laboratory—Markus, Thomas, Axel, Dirk—and ZigPos GmbH—Erik, Christoph, Tino—who together form a complete and highly professional research team.

My second internship took place at Delft University of Technology, under the guidance of Alexandru Iosup. I am grateful to Alex for many things, such as keeping close contact with UPB, maintaining an orderly yet creative perspective on our Mobile Social Gaming project, reviewing the drafts for countless times, and teaching me box plots and basketball. The PDS group, led by Henk Sips and Dick Epema, forms a great working group, both professionally and socially. I would like to thank Otto Visser, Siqi Shen, Yong Guo, Kefeng Deng, Lipu Fei, Bogdan Ghiț, Mihai Capotă, Boxun Zhang and the whole PDS Group. We also had a fruitful collaboration with Fernando Kuipers and Emanuel Dias from the Network Group. I hold dear the company of my friends in the Netherlands, among whom I also count, in addition to the aforementioned: Andra, Andrei, Bogdan N, Ana, Marius, George, Jie, Adele, Mihai T, Ionuț, Andreea, Erdal, Manu, Oana, Lucas and Niek.

Last, but certainly not least, I would like to thank my friends from home, for their refreshing support Ștefan, Laura, Dan, Roxana, Petre, Ruxi, Vali, Andreea N, Vlad, Irina, Ioana, Țache, Măda, Anda, George, Anca and Adi. Foremost, I would like to thank Eluș, mom, dad and the rest of my family, for being close to me even when I was far away and for their continuous support. I love you all very much!

Alexandru Olteanu

Abstract

Modern mobile devices, such as smartphones and tablets, are offering increasingly complex functionalities. We use them daily in activities ranging from entertainment, to solving professional tasks. Although mobile devices are growing in functionality and computing power, we believe the role of more powerful infrastructure, to augment the capabilities of mobiles, will increase. Surveying the state of the art, we can see that developers and researchers alike study ways to access from the mobile device resources in the Cloud, in community resource pools, and in personal networks.

The focus of our work is to define an exploratory space for offloading concerns and to use it in the analysis and empirical evaluation of various offloading mechanisms for mobile devices. *First*, we survey existing research efforts regarding offloading for mobile devices and propose a General Offloading Model and a Taxonomy. We also propose an Exploratory Space that shows novel offloading mechanisms, as well as the opportunity of a design space exploration. *Second*, we propose a workload model for online social applications, for both macro-level information—the number of users—and micro-level information—the operations users trigger. We collect traces for thousands of Facebook games and several native mobile applications, and use it to characterize and model various elements in the workload model. We also show how our model can be used to generate synthetic traces and to help developers understand how the changes in their applications can affect the workload. *Third*, we conduct a comparative analysis on offloading mechanisms for various mobile applications. We investigate Communication Adaptation and Offloading for applications spanning the mobile device and custom hardware extensions, and Computation Adaptation and Offloading for loop-based applications. We also propose an operational analysis to describe offloading mechanisms for loop-based applications that, according to our workload model, include online social applications. *Fourth*, we conduct performance evaluation on various offloading mechanisms. We show results for empirical evaluation on a real-world application, a simulation that estimates performance for the whole population of users and an analytical evaluation to match our proposed operational analysis.

Offloading for mobile devices is a hot topic and will continue to be, as people are surrounded by seamlessly interconnected personal computing devices. We found a strong community interest for our workload model, leading us to plan further improvements, that will refine the model and allow it to cover more applications, with greater accuracy. We also believe that our comprehensive exploratory space can continue to serve as basis for further empirical evaluation, both with real applications and simulations.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Modern mobile devices, such as smartphones and tablets, are offering increasingly complex user experiences. We use them daily in activities ranging from entertainment to solving professional tasks. As sales of mobile devices surpass personal computers sales, matching advances in hardware and mobile computing with the requirements posed by this massive mobile momentum is challenging for developers and researchers alike. Although mobile devices are growing in functionality and computing power, we believe the role of more powerful infrastructure, to augment the capabilities of mobiles, will increase. Thus, this PhD research focuses on techniques for offloading operations from mobile to more powerful cloud-based infrastructure.

1.1 The Context of Cloud Offloading for Mobiles

Modern handheld devices, such as smartphones and tablets, offer portability, increased computational power, and communication capabilities. Thus, they are becoming an attractive option for users to interact with each other, through social applications, and with their environment, through ubiquitous computing[1]. Facebook, who has announced recently their increase to over 1 billion monthly active users, reports that more than a half of the users reach their social network using a mobile device [2]. On the other hand, along with the technological advances in hardware and mobile computing, user demands are also increasing, as they expect content rich applications, like games, and access to large amounts of remote data, like multimedia streaming. Advanced as they may be, modern mobile devices still have some limitations in relation to user demands, in terms of battery supply, memory capacity and heat dissipation. Thus, it is reasonable to see why mobiles, despite their increasing computing power, continue to use more powerful infrastructure.

The convergence of mobile and cloud computing has been studied for a number of years and is still a hot topic, because of the dynamics in mobile computing and because of the challenges that continue to arise. As sales of mobile devices grow above sales of personal computers, many hardware and software manufacturers compete on the mobile market. Companies such as Samsung, Nokia, HTC, Motorola, Apple, Acer and Asus produce mobile devices of various hardware characteristics, using a variety of operating systems, either developed in house, like Apple's iOS, or by large software companies, like Google's Android or Microsoft's Windows. The heterogeneity of mobile devices, in terms of hardware and operating systems, makes it difficult for application developers to reach all the mobile users, and they usually have to maintain several versions of the same application. Cloud providers are also interested in tuning their systems to face big variations in the number of users. In this massive ecosystem, researchers find challenging topics with effects ranging from user experience to cost optimization for the resource providers.

Cloud offloading is one of the emerging trends in distributed computing involving mobile devices. Developers and researchers alike study ways of accessing, from user terminals, the power offered by cloud infrastructure in terms of storage. The cloud has been used for offloading storage and functionality for computers for a long time. However, more recently, mobile devices encouraged developments in computation and communication offloading, with a focus on the trade-offs between the benefits that the powerful infrastructure brings and the costs, in time and money, of using remote resources.

1.2 Problem Statement

Our vision is to seamlessly interconnect mobile devices, as part of a hierarchy of computing devices in relation to their proximity to the user. This hierarchy, which we discuss in detail in Section 2.1, is essentially comprised of wearable devices, handheld devices, local infrastructure, and clouds. Latency is smaller in the proximity of the user, but resources are more powerful further away. It is thus desirable to find a balance between the trade-offs that govern this hierarchy.

The convergence of mobile and distributed computing has been studied for a number of years, with results in system design[3][4], job scheduling [5][6], resource discovery [7][8] and so on. Mobile integration with various other types of computing takes many forms, like mobile cloud computing, offloading, delegation, cyber foraging, and data staging. We focus on offloading, which is a form of transferring tasks of various granularity to remote resources. Offloading and delegation are very similar approaches to use remote resources. They are sometimes considered to be complementary, as in the work published by Flores [9]. Our work focuses on offloading, which is detailed in the next section. Cyber Foraging is an opportunistic approach of using remote resources from mobiles. Satyanarayanan [10] introduced this concept in 2001 as a pervasive computing technique, and work within the same research team [11] led to a scripting language for cyber foraging. Verbelen et al. [4] introduced AIOLOS, a middleware to improve mobile application performance through cyber foraging and Kristensen [12] introduced scheduling concepts in the topic.

Several offloading systems have been proposed, as middlewares, frameworks, or services. However, we find that few solutions reach the point to have a big impact on live systems and applications. Studying mobile offloading is challenging because of device and application heterogeneity. However, we believe that focusing on a specific type of application can bring advances in offloading for mobile devices, while still keeping a wide range of applicability. Thus, our approach is to conduct an application domain exploration and select a family of applications on which to conduct analysis and evaluation of various offloading mechanisms.

1.3 Main Challenges and Research Questions

Offloading for mobile devices is an increasingly popular research topic, matching the popularity mobile devices have in the general population. With the first research efforts targeted specifically on mobile devices dating back in the 1990's, in the past couple of years a vast material on offloading for mobile devices has been published. Our *first challenge* is to cover this vast material, organize it and find the opportunity for our novel contribution.

Many of the modern mobile applications are designed for users to interact with other users, e.g. social platforms like Facebook, or to share information, e.g. photo sharing platforms like Instagram. Moreover, most mobile platforms have a market for mobile applications, like Google Play or Apple iTunes, where users can also interact when deciding to download applications. Thus, mobile application usage is greatly influenced by relations among users and the *second challenge* we are facing is to analyze user behavior and workload in the context of user social interactions.

Software that uses the interaction of mobile devices with the cloud is already on the market. However, recent research materials have identified the cloudlet as an offloading target [10], emphasizing the trade-off between communication and computation costs. Inspired by this research, we see mobile devices as part of a hierarchy of computing systems people are using today, which is essentially comprised of wearable devices, handheld devices, cloudlets, and clouds (see Figure 2.1). Given the rate at which smart devices become smaller and closer to the user, we find it foreseeable that in the near future mobile devices will serve not as sources, but as targets for offloading, from smaller, wearable devices. The *third challenge* we are facing is to decide how to conduct the offloading process, in the context of online social applications.

The recent popularity of smart mobile devices is motivating many manufacturers to produce devices for many types of consumers, thus leading to orders of magnitude in heterogeneity. The processing unit may vary from single-core CPU, to quad-core CPU, and even to hybrid architectures that include a dual-core CPU and a GPU. The battery lifetime may vary from tens to hundreds of hours in standby, and is greatly influenced by user behavior, as intense device usage can reduce its battery life to barely a few hours. The *fourth challenge* we are facing is to assess the benefits of offloading under such heterogeneity.

To match these challenges, we propose these research questions:

- What is an exploratory space of offloading concerns for mobile devices? Which novel offloading mechanisms could be beneficial for mobile devices?

- How to characterize and model the workloads for mobile applications that benefit from offloading, given that the social interactions among users usually have great effects on the workloads of this type of applications?

- What mobile application type is suitable to study offloading mechanisms? How to measure the improvement that a mix of offloading mechanisms can bring for various devices under realistic workloads?

- What improvements can various offloading mechanisms have for various devices under real and synthetic workload traces?

1.4 Main Objectives

This thesis focuses on defining an exploratory space for offloading concerns and using it in the analysis and empirical evaluation of various offloading mechanisms for mobile devices. To accomplish this, we need to structure the vast information from the research literature on offloading for mobiles, to investigate how various offloading mechanisms work on several types of applications and pick an application domain for which to show in detail a selection of offloading mechanisms from our exploratory space. Thus, the main objectives of this thesis include the following:

- study and organize the vast amount of aspects about offloading for mobile devices, for identifying novel offloading mechanisms and for defining an exploratory space
- characterize and model workloads of a comprehensive number of applications, taking into consideration the social interactions among mobile users, which usually have great effects on the workloads
- investigate in detail how various offloading mechanisms work on a few applications and derive an analytical support that can be extended on a whole family of applications
- use the workload traces and the analytical support to conduct analytical evaluation on the offloading mechanisms in the exploratory space, and validate the findings through empirical evaluation on a real-world application.

1.5 Thesis Structure

In this section we present the thesis structure and a brief overview of how we address our objectives and how we accomplish our contributions summarized in Section 6.1.

The thesis is structured in four chapters, each one addressing one of the four objectives of this work. Figure 1.1 summarizes the relation between the four chapters, and the remainder of this section details it.

In Chapter 2 we survey existing research efforts regarding offloading for mobile devices and propose a primer on offloading, a General Offloading Model and a Taxonomy for offloading concerns, to structure the vast possibilities of conducting the offloading process. We also identify two main research directions. First, we emphasize the balance between offloading and alternatives, such as adaptation, and define three main factors that rule this balance: application type, offloading type and offloading tuning. Second, to structure our study on these three factors, we propose an Exploratory Space that outlines a design space exploration, as well as the opportunity to indicate novelty in offloading mechanisms, such as parallel and partial offloading.

In Chapter 3 we summarize our efforts to collect traces of online social applications that contain information about workloads at macro-level (number of users) and at micro-level (operations) for 16.000 Facebook games and several mobile applications. We also propose a workload model with several components: Popularity Distribution and Evolution Pattern, for macro-level information, Packet-Level Statistics and State Transition Diagram,

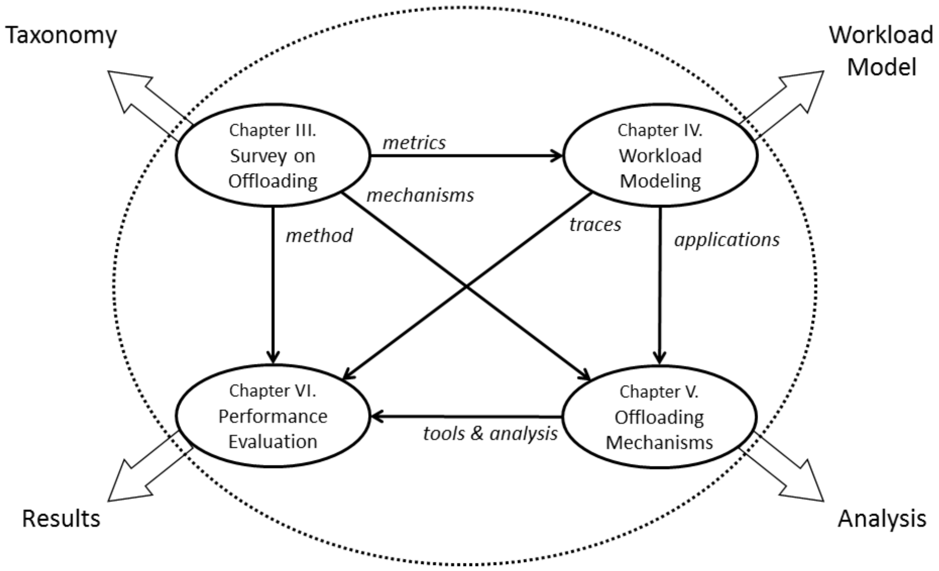

Taxonomy
Workload Model
Chapter III. Survey on Offloading
metrics
Chapter IV. Workload Modeling
mechanisms
method
traces
applications
Chapter VI. Performance Evaluation
tools & analysis
Chapter V. Offloading Mechanisms
Results
Analysis

Figure 1.1: The relation between the main four chapters of the thesis.

for micro-level information. Then we present the results we obtained from characterization and modeling using the traces of online social applications on the workload model, showing that normal applications, with a single peak in user numbers, reach maturity early in their lifetimes, and that social networking features are increasingly influencing the user experience of online applications. Finally, we show how our workload model can be used to predict traffic and evolution typology, as well as to generate synthetic traces.

Chapter 4 shows our efforts to investigate in detail some offloading mechanisms on a few real applications. We study Communication Adaptation and Offloading for distributed applications spanning the mobile device and custom hardware extensions, such as sensor devices and home automation networks. We also study Computation Adaptation and Offloading for loop-based applications, that is applications which function by iterating a processing loop, with a focus on video processing, using an augmented reality application, and video rendering, using a popular simulation game. We then propose a formalism to help conducting operational analysis for the whole family of loop-based applications, for the mechanisms that form the Exploratory Space we defined in Chapter 2.

In Chapter 5 we reiterate through the mechanisms that form the Exploratory Space from Chapter 2 to conduct performance evaluation. We conduct empirical evaluation using a testbed that we designed and implemented based on a real world application, and analytical evaluation using the formalism we proposed in Chapter 4 and the workload traces from Chapter 3. We show the results and compare them for validation.

Chapter 6 concludes the thesis by stating how this work fits among the current efforts on offloading for mobile devices. We summarize the main contributions of our work, such as defining an exploratory space for offloading mechanisms, modeling the workloads of thousands of applications, investigating in detail how several offloading mechanisms function on a few applications and how this functionality can be generalized for a whole family of applications through operational analysis, and validating the analytical results with empirical evaluation. Finally, we describe the future directions opened by our work, such as improving the workload model for better accuracy and for more applications, providing a more complex analytical model to better formalize how tasks are handled in an offloading system, and extending the workload predictions on the computational infrastructure with possible results in capacity planning and provisioning.

Chapter 2

Aspects about Offloading for Mobile Devices

To accomplish our vision, of seamlessly integrated mobiles within the computing world, we have studied the existing literature and found there are several approaches, such as offloading, delegation and cyber foraging. We focus on offloading as an approach with many opportunities in novel research. In this chapter, we survey existing research efforts regarding offloading for mobile devices and propose a Primer on Offloading, a General Offloading Model and a Taxonomy for Offloading Concerns, to structure the vast possibilities of conducting the offloading process. We also identify Research Directions: we emphasize the balance between offloading and alternatives, such as adaptation, and define three main factors that rule this balance: application, offloading type and offloading tuning. To cover multiple aspects of the latter, we propose an Exploratory Space that outlines a design space exploration, as well as the opportunity to indicate novelty in offloading mechanisms, such as concurrent and parallel offloading, spatial and temporal partial offloading, and temporal content offloading.

2.1 A Primer on Offloading

In this section we provide an elementary material on offloading, to serve as introduction to the broad subject of offloading. We provide a motivation, main goal, characteristics, benefits and challenges. Throughout the section, we refer to the most important aspects that lead to offloading for mobile devices: mobile characteristics, their popularity, the vast application domain of mobiles, the way they interconnect with a more powerful infrastructure and how this connection is structured.

2.1.1 Offloading Motivation

Modern handheld devices, such as smartphones and tablets, offer portability, increased computational power, and communication capabilities. Among their *characteristics* we outline several that we find most important:

- Connectivity: mobile devices are enabled with WiFi and 3G modules to connect to the Internet, and with Bluetooth, NFC and various serial interfaces to connect with their peers and with dedicated devices. Communication protocols are continuously evolving to offer higher bandwidth and lower energy consumption.

- Processing capabilities: encouraged by the competition among mobile device manufacturers and user demand, processing capabilities are growing, closing the gap with personal computers. High-end mobile devices feature quad-core CPUs and even GPUs. However, limited by energy concerns, all processors employ frequency scaling and usually offer high performance for limited periods of time. Moreover, user demand is growing beyond the capabilities of single devices, in applications such as games.

- Sensing abilities: all mobile devices have a number of built-in sensors, such as accelerometer, gyroscope, compass, pressure meter, luminosity, and input functions, such as touch, voice commands, that enable them for a variety of applications. Moreover, through Bluetooth and serial interfaces, they can connect to dedicated sensing devices.

- Pervasiveness: handheld devices are designed for being close to the user at all times. They become personal assistants, or gateways in the interaction with other devices. Besides their portability, they are more attractive than personal computers because they are always on, and thus offer lower latency at boot time. The tradeoff is their small display.

- Heterogeneity: the popularity of mobile devices, encourages numerous manufacturers to enter the race for market. Each offers its own platform. Far from being the only operating system on the market, Android is a step towards unifying their efforts, but still each manufacturer has to customize the Android implementation. The need for unification is sprawling technical solutions such as HTML5, that offer the possibility of running the same code on multiple devices under the responsive paradigm, but it is far from offering all the functionalities a native application can access.

- Limited battery supply: this is probably the characteristic that triggers most discussions. While some complain about the limited battery capacity most devices have, others appeal to bigger batteries that affect the form factor, while others are satisfied with daily recharge cycles. Overall, battery supply is directly affected by the performance of the device and always will be a tradeoff to it.

The limited battery supply and processing capabilities, at least in respect to the user demand, are the characteristics that trigger the most interest in offloading research. The connectivity supports the offloading process, while the heterogeneity provides several challenges.

Given the characteristics mobile devices posses, they are an attractive option for users to interact with each other, through social applications, and with their environment, through home automation. The *popularity* of mobile devices can be seen in many ways. Facebook, who has anounced recently their increase to over 1 billion monthly active users, reports that more than a half of their users reach their social network using a mobile device [13].

People use mobile devices daily in activities ranging from entertainment to solving professional tasks. Mobile applications span *a vast application-domain*, being developed for various purposes, such as gaming, multimedia streaming, travel, communication, etc. Many of these types of applications rely on connectivity and on data stored remotely. Also, many of them make a lot of use of the high computation power of mobile devices. Among this generous application space, there are several type of applications that would benefit from offloading:

- applications that are computational intensive, like Chess and other low-response-time games: these applications are fit for remote processing because users do not expect quick responses from the program
- applications that rely on data from server side, such as Shazam, the program that recognizes songs by comparing samples with a vast cloud database: the amount of data these applications use is prohibitive to a single device functionality
- applications with pipeline functionality, such as image processing applications: these are suitable for offloading in various degrees, as only parts of the pipeline should be offloaded, depending on the conditions at hand
- applications that already interact with the cloud, such as m-commerce applications: these are already making use of the communication capabilities, so employing other forms of offloading is just a matter of fine-tuning

Although mobile devices are growing in functionality and computing power, we believe *the role of a more powerful infrastructure*, to augment the capabilities of mobiles, will increase. First, there are still limitations mobile devices have due to battery performance and power dissipation. Second, personal computers and large computing clusters are also growing in power, so there will always be a gap between the best mobile device and the best computing station. Third, the requirements users put on devices are continuously increasing, triggering software and hardware development cycles. Fourth, an increasing number of applications rely on global data, either because this data is too big to store on a single device, or simply because it comes from multiple sources, like crowdsourcing or social interactions.

Software that uses the interaction of mobile devices with the cloud is already on the market. However, recent research efforts have also identified the cloudlet as an offloading target [10], emphasizing the trade-off between communication and computation costs. Inspired by this research, we see mobile devices as part of a hierarchy of computing systems people are using today, which is essentially comprised of wearable devices, handheld devices, cloudlets, and clouds (see Figure 2.1). Given the rate at which smart devices become smaller and closer to the user, we find it foreseeable that in the near future mobile devices will serve not only as sources, but as targets for offloading, from smaller, wearable devices, such as glasses. Figure 2.1 also shows the tradeoffs encountered in this hierarchy of devices: devices closer to the user are connected with better, low range communication methods, that ensure lower latency, but also generally have less resources. Overall, there is a need to minimize costs and energy consumption.

The convergence of mobile computing and cloud-based services is one of the leading ways in which cloud computing is evolving [14]. As the Forrester analyst, Glenn O'Donnell, put it: "‘cloud plus mobile is a classic more than the sum of its parts combination"’.

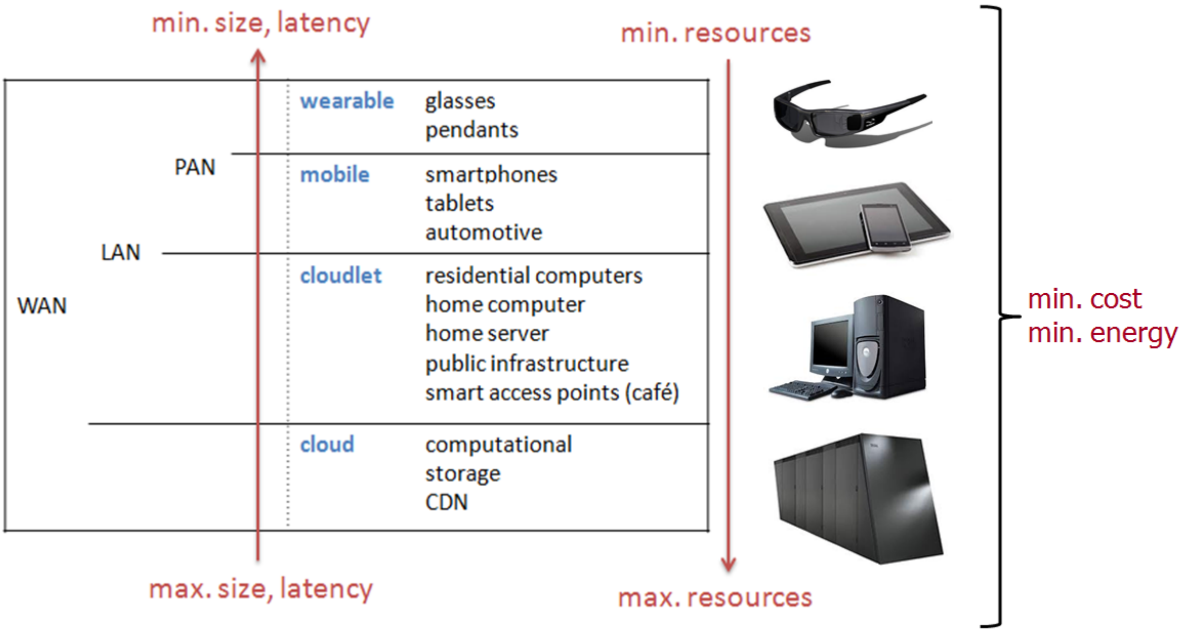


Figure 2.1: Hierarchy of devices in relation to their proximity to the user. Interconnection level: *Personal Area Networks* (PAN), *Local Area Networks* (LAN),*Wide Area Networks* (WAN). Trade-offs are highlighted.

Mobile integration with various other types of computing takes many forms, like mobile cloud computing, offloading, delegation, cyber foraging, and data staging. We focus on offloading, which is a form of transferring tasks of various granularity to remote resources. Several offloading systems have been proposed, as middlewares, frameworks, or services. In this work, we try to find and formalize the patterns underlying various types of offloading systems.

2.1.2 Main Goal

The goal of offloading is to use remote resources to improve the functionality of the source device. Offloading can be done in order to either maximize performance, to minimize energy, or to minimize cost. Usually, these goals are mutually exclusive, as, for instance, better performance means more power consumption, or higher costs. Another interesting tradeoff when offloading is between the gain on computing power, due to more powerful infrastructure, and the loss on data transmission, due to the distance to the remote resource. Moreover, offloading decisions need to be made in a setup that is continuously changing, due to device mobility, fluctuating connectivity, and variating energy supply.

When properly conducting offloading, it is important to identify the appropriate conditions in which offloading can bring a benefit and to operate within given parameters. The vast application domain, the device heterogeneity and the complex process of offloading offer researchers the possibility to investigate various forms of the offloading process in a number of conditions. Proper mobile device, having certain types of workloads, for certain applications, using appropriate infrastructure, under specific conditions are all key elements in conducting a successful and beneficial offloading process. Moreover, assessing these conditions accurately, automatically and as quickly as possible is the main goal of the research in this domain.

We further investigate the main goal from the perspective of several research approaches in Section 2.2.3.

2.1.3 Offloading Characteristics

Offloading aims to optimize the functionality of an application by using remote resources. Although most of the existing research efforts focus on key concerns, such as performance, energy and cost, some of them acknowledge that offloading is much more complex, especially if it is designed to work with a real-world application. Inspired by research on distributed systems in general, we have in mind a broader range of concerns:

- *Performance* is, of course, the main concern of an offloading system. In order to offer an increased performance to its clients, the offloading system should be fine tuned to have its own performance at peak efficiency. Performance analysis is a complex task, but all of its aspects, ranging from modeling to measuring, have been researched and applied on various distributed systems.
- *Energy saving* is key for all modern computing systems. Mobiles are focused on energy saving due to their limited battery supply. Clouds are also focused on energy saving to ensure low costs for their users.
- *Costs*, from a financial point of view, can also become a complex aspect of offloading. A single offloading operation can imply costs for multiple service or resource providers, such as network operators, software manufacturers and cloud owners. The cost for data transmission can range from nothing, for WiFi, to a significant amount of money, if 3G/4G is used. Remote processing can be free if some local infrastructure is used, or can cost a fee, usually in the pay-as-you-go form, to service providers such as Amazon Web Services or Microsoft Windows Azure.
- *Accuracy of the results* can also be a serious concern, especially when processing is done in parallel on the device and on a different architecture, with various types of data
- *Scalability* in offloading systems, as in any distributed system, is a serious concern when addressing large numbers of inputs. For example, if the offloading system goes public, it needs to scale well to ensure proper functionality for increasing numbers of users.
- *Elasticity* is the ability to adapt to workload changes and it usually involves actively creating and destroying resources. Thus, the system should either predict or react quickly to both positive and negative changes in the workload. If the system is not able to provision new resources, the clients will be affected by lack of service. If the system is not able to deprovision unused resources, then the financial cost will grow unnecessary for existing clients.
- *Flexibility* refers to the property of a service to be customized to better serve the needs of various types of customers

- *Support for value adding operations* like backup, update, cloning and avoid vendor lock-in
- *Security* is, like in any distributed system, a topic of great interest, because data leaves the personal device and needs to travel over public infrastructure for remote processing. Thus, there are many existing and emerging solutions on security that can be applied in offloading systems as well.
- *Support for regulatory law*, which includes privacy and durability, is also a key aspect of offloading systems. People are usually sensitive about the data they keep on their mobiles, which are used as personal assistants. Durability is a mirror aspect of this concern for data, which should not be lost unless the user requires such a deletion.

Among offloading characteristics, we can also number:

- setup-related characteristics: what resource will be offloaded and how will it be determined
- execution conditions: which is the target of the offloading, what is the allocation policy, what are the system properties, and so on
- how will offloading be operated: which metrics control the process, which offloading mechanism will be used

We detail all of these characteristics in our General Offloading Model in Section 2.2.

2.1.4 Offloading Benefits

Offloading has a number of benefits, some already exposed in commercial applications, other still only shown in research studies.

Offloading addresses some of the limitations of mobile devices. For example, to prevent mobiles from performing numerous queries to services, the major mobile operating systems developers implemented push notification services, a form of communication offloading. Data and content offloading opened the way for feature recognition applications such as Shazam, that rely on massive amounts of data, that could not exist on a single device. Moreover, offloading, as a centralized service, addresses all the types of devices, which, as we have seen, reach orders of magnitude in heterogeneity.

Another benefit offloading systems have is that they can service a large amount of users, using the already validated examples of web-based applications. In web-based applications, a scalable amount of users reach a remote resource using their browsers. Offloading can, therefore, address users on a global scale, and they can easily understand how it works, because they can relate to these similar models.

From the developer's perspective, besides increasing performance, offloading can ease the development process, because it follows the one implementation for all model. Offloading also moves processes closer to the developer, who knows how to fix issues and how to update features, which leads to better service times.

Offloading can be used for various purposes, to increase performance, to enable new functionality on mobile devices, or to enable new properties in mobile applications (like fairness in games). Offloading can also make it feasible to produce wearable devices, smaller, less capable devices, focused on a single function, like collecting statistics for joggers. Such devices can use mobiles as their offloading target.

2.1.5 Main Challenges

Mobile characteristics have a great impact on the offloading process. Mobility leads to wireless connection losses and handovers which can pose some technical difficulties. Communication in mobiles also poses a challenging tradeoff, between WiFi, which has more bandwidth and less cost, and 3G, which is pervasive. Mobiles have sensing abilities, like location awareness, that might lead to concerns regarding data sensitivity and privacy. Also, device heterogeneity, in terms of hardware and technologies used by OS manufacturers, are a serious challenge to any offloading system, that essentially needs to replicate functionality so it can perform with any kind of device.

The popularity of mobile applications also poses interesting challenges. Mobile applications have peculiar workloads, with social influences and bursty behavior, that require good provisioning mechanisms. The simple use of traditional thin client approaches is not sufficient in many cases, so better tuning is needed.

Studying mobile offloading is challenging because of device and application diversity. The recent popularity of smart mobile devices is motivating many manufacturers to produce devices for many types of consumers, thus leading to orders of magnitude in heterogeneity. For example, the processing unit may vary from single-core CPU, to quad-core CPU, and even to hybrid architectures that include a dual-core CPU and a GPU. The same level of diversity is found in mobile software as well, due to the massive developer population that builds and maintains hundreds of thousands of applications on several official marketplaces, like Apple iTunes, Google Play or Microsoft Windows Store, and countless applications on unofficial marketplaces. Mobile applications span over various genres and implementation technologies. For worthwhile offloading, it is necessary to find those applications where offloading is beneficial. Most offloading efforts focus on finding a good ratio between computation and communication costs. We focus on applications that already have some sort of communication, so the challenge is just fine-tuning this communication. Instrumentation is key in performing such fine-tuning and it usually starts with profiling, either online (interval-based, event-driven, etc), or offline.

Offloading implies the usage of third-party infrastructure in an interactive approach. It is therefore necessary to quickly decide if offloading is beneficial or not. Such a decision, as well as allocation details, can be made through either cost-driven (how much it costs in energy and time?), or data-driven (where is the data?), or code-driven (where is the code?) analysis [15]. Moreover, security and multi-tenancy issues must be taken into consideration.

Finally, because mobile devices are part of a hierarchy of devices, offloading requires to quickly decide where to offload in this hierarchy of devices and to account dynamic changes of context. For example loosing access to a network, or the battery level getting below a threshold, can have dramatic effects on the benefits of the offloading process.

In our work, we try to address these challenges, look beyond the surface to the underlying

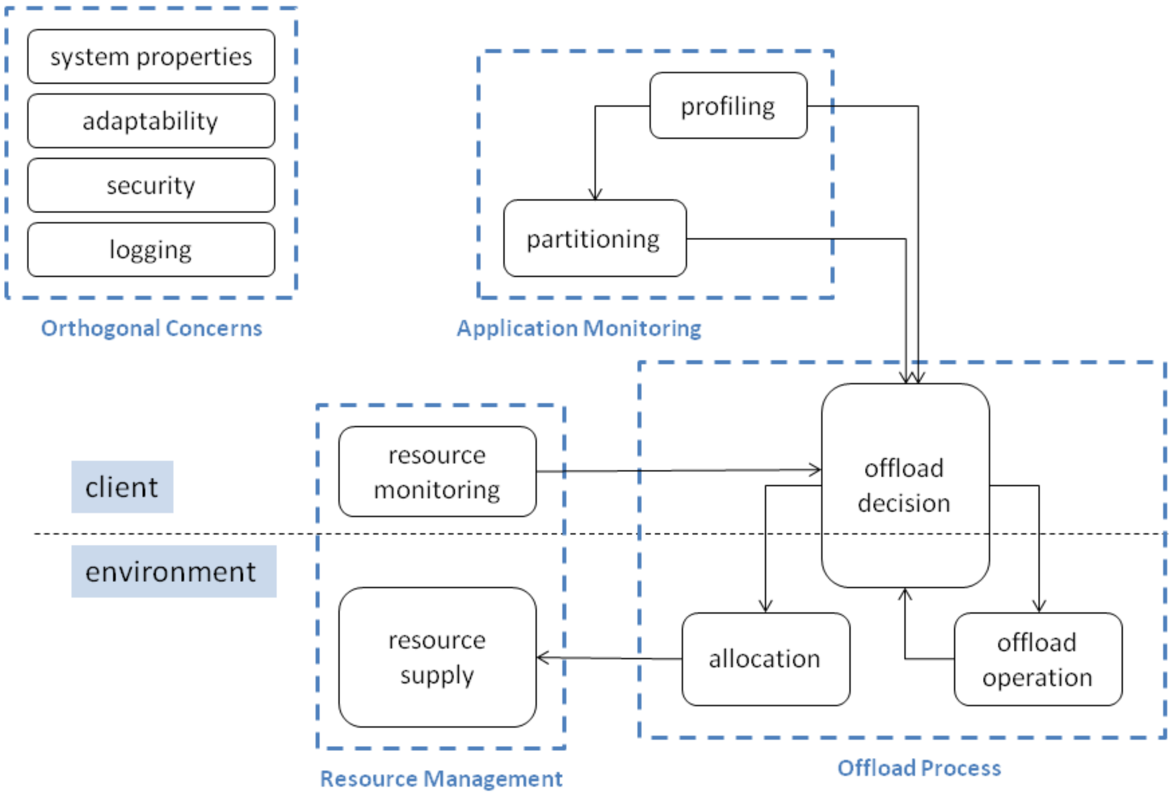


Figure 2.2: General offloading model.

technology and to find those mechanisms that span a broad range of applications, but still offer the possibility of some in-depth insights.

2.2 General Offloading Model

In this section, we present a general offloading model that describes a generic offload system. We will then map the various research efforts as subsets of this general model. In this chapter we offer an overview of each of the key components involved in offloading and we will detail their functionality in Chapter 2.3.

We divide the components of an offloading system into two planes: components on the client—the mobile device—and components in the environment—either a cloud, a cloudlet, a peer device, or a hybrid environment, as discussed in Section 2.1.1. The components are depicted in Figure 2.2 and are detailed in the remainder of this section.

Many of the current research efforts focus on thoroughly understanding the application to be offloaded. Therefore, the *Application Monitoring* part of the offloading system is key in obtaining a beneficial offloading process. The *Profiling* component is able to assess the way in which the application is functioning through various mechanisms, such as static or dynamic analysis. The information obtained from the Profiling component may be used by the *Partitioning* component, which aims to split the application in components of predefined granularities and identify which of them are offloadable. Both the Profiling and Partitioning components provide information essential to the offloading decision.

There is also a need to assess the resources that are available for the functioning of the application, both local and remote, with a *Resource Management* component that spans both the client and the environment plane. In the client plane, the *Resource Monitoring* component assesses parameters such as battery level, CPU load, wireless connection quality and so on. In the environment plane, the *Resource Supply* component manages the external resources that may be used in offloading, through mechanisms such as discovery and provisioning. Resource discovery is useful in opportunistic approaches, such as cyber foraging, in which the mobile device tries to find available offloading targets in its environment. Resource provisioning is a more proactive approach, highly utilized in cloud environments, in which resources are dynamically created to adjust to computing needs.

The *Offload Process* itself needs to be an iterative process, due to mobility and the changing nature of the execution conditions. For example, the mobile client may switch from WiFi to 3G, or may reach a critical battery level, with consequences the offloading process. The *Offload Decision* component receives information from Application Monitoring and Resource Management, to assess the current offloading needs and conditions, and from previous iterations of the *Offload Operation*, to assess their benefits and defects. Offload Decision can choose:

- *what* to offload, among the sets of partitions offered by the Partitioning component
- *when* to offload, as sometimes it may be not worth offloading at all
- *where* to offload, and instruct the *Allocation* component to use the appropiate offloading target.

Besides the basic offloading process, research efforts also address a number of *orthogonal concerns*. Some approaches focus on *adaptability*, e.g. a game may turn off animations if the offloading system is not able to sustain a reasonable frame rate, or the device may switch communication networks through handover. *Security* concerns derive from using remote resources, that may belong to third-parties, during the offload process. *Logging* may be used for accountability, billing and making software improvements.

In the remainder of this section, we provide a mapping of existing research efforts on these components. In the following section we provide a taxonomy based on the components presented here.

2.2.1 Application Monitoring

Some of the researchers [10] [16] [17], focus their efforts on one or a few applications, for which the behavior is considered a priori knowledge. Most of the research efforts [18] [19] [20] [3] [21] employ a mix of static and dynamic analysis mechanisms. Notably, there are a few efforts [22] [23] [24] [4] [3] that employ the type of profiling that monitors the application behavior while offloading. Also notable is code instrumentation [23] as a profiling method, that steadily evolved into more automatic ways [18].

Also in the application monitoring area of concerns, partitioning refers to dividing the application into components of various granularities, to obtain a graph. Some researchers quantify each node and each edge with specific metrics and apply clustering algorithms to obtain a partition or set of partitions [3] [21].

2.2.2 Resource Management

Some approaches focus on challenges regarding resource handling, both locally and remotely. Locally, Resource Monitoring assesses resources available on the device, most often CPU and network [11] [25].Remotely, Resource Supply needs to assess resources available in the environment, which can either be performed by discovering or provisioning resources.

Zhang et al [26] focus on two scenarios that require massive resources in constrained locations, namely disaster relief and child finding. This type of scenario encourages the use of devices in the vicinity, by leveraging collocation. In [10], the authors describe the Kimberley Control Manager, that supports browsing and publishing services using the Avahi mechanism in Linux. This module acts as a broker in identifying and maintaining connections with the remote resources. In the work presented by Marin et al [27], a component named Cloud Receiver also acts as a broker in deciding which cloud resource to use.

Ferber[28] and Satyanarayanan[10], who use virtualized environments for conducting offloading, measures the demand and provision when necessary, taking into account power consumption and startup overhead. Also notable is the solution proposed by Yan[29] that also considers resource specialization when provisioning. At the same time, the existing solutions having an opportunistic approach [26] do not employ any provisioning at all.

2.2.3 Offload Process

The offload process needs to be an iterative process, to react to changes in the running conditions. The iteration involves deciding on the form of offloading, allocating resources and performing the offload operation.

Most researchers take into account the performance of the mobile device with and without offloading, expressed as running times. Some also optimize the energy consumption, like [30][31][26], or the monetary costs, like [28][29][26]. Satyanarayanan[10] and Verbelen[20] also refer to the quality of the result when offloading, like better image resolution or more accurate matchings. Usually, the researchers adopt a strict, mathematical assessment model, based on equations. However, in [32], the authors present a prototype based on fuzzy logic that takes into account remote information sources. They demonstrate their prototype by uploading videos and applying a map-reduce version of a face detection algorithm.

The usage of the remote resource produces an output that needs to be merged in the local processing. This is sometimes a challenging task, as it implies parallel programming concerns, such as synchronization, concurrency and error handling. Most of the solutions based on remote code execution discuss methods to integrate the results obtained on the remote resources back into the mobile device. Some solutions, based on virtual machines, like [10] and [4] only discuss state integration. Notably, some approaches, like [33][29][15] get into the complexities of handling failures in remote processing. The easiest solution is to acknowledge failures through keep-alive mechanisms and to correct result errors by discarding all the results that come from a faulty remote resource.

2.2.4 Orthogonal Concerns

We classify system properties as an orthogonal concern, because they refer to the system as a whole. The performance property of the offloading system is discussed by all solutions we studied. Availability, durability and reliability are also discussed quite often [18][34][35][36] as these are key properties when trying to develop an offloading system beyond the prototype phase and into a public solution. In addition, systems such as the ones presented in [34] and [37] also discuss the distribution property of the system, while others, especially the ones using cloud infrastructures, leave that to the cloud service provider.

Kumar et al [30] discuss several privacy concerns related to using cloud resources, giving examples such as bugs, third-party vendors and location tracking. They identify as possible solutions encryption and steganography. Eom et al [18] propose to use SocialVPN, a tunneling technique through Virtual Private Networks that has a double benefit: better security and virtualisation. Kemp acknowledges [36] security and privacy concerns users might have when offloading, but leaves them for future work.

Given the research nature of the materials we have studied, most approaches are at a prototype phase and only consider logging for functional and debugging purposes[34]. However, it is easy to understand how commercial offloading solutions like Google C2DM [37] and, more recently, GCM, provide logging for accountability and billing purposes.

2.3 Taxonomy for Offloading Concerns

In this Section, we present a novel taxonomy for offloading concerns, structured on the model presented in Section 2.2. We first introduce our taxonomy and then provide a mapping with the research literature we studied on offloading for mobile devices.

Some approaches focus on the partitioning part of the process, discussing ways to instrument and to profile the code. Other approaches focus on the discovery of resources.

2.3.1 The Taxonomy

As shown in Section 2.2, we identify four major areas of offloading concerns: Application Monitoring, Resource Management, Offload Process and Orthogonal Concerns. Each of these areas has a number of key topics of interest, each with several subtopics. Figure 2.3 shows our three-level taxonomy.

A. Application Monitoring

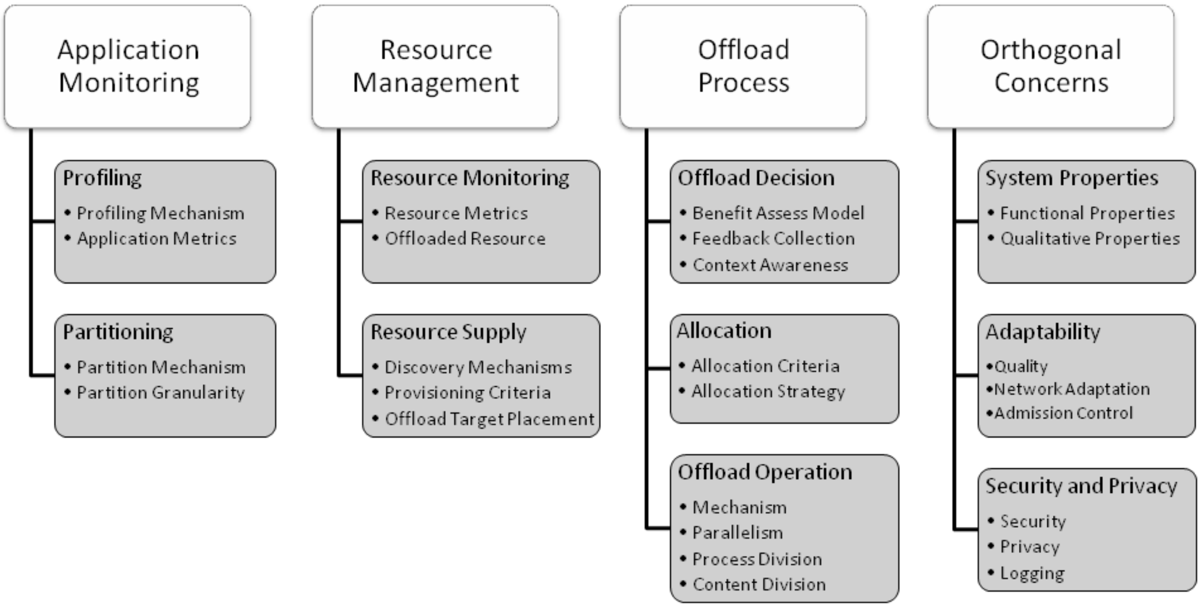


Figure 2.3: Our Taxonomy for offloading concerns.

As shown in Section 2.2.1, Application Monitoring is a key area of offloading concerns, as it helps to decide what part of the application is worth offloading and under which conditions.

A.1. *Profiling* is a classical approach in application optimization, usually employed by developers when optimizing the functioning of an application, e.g. when implementing a parallel version of a serial program. Profiling is also important when offloading, offering input for both partitioning and offloading decision.

A.1.1. *Profiling Mechanism.* Profiling can be accomplished through various mechanisms. Static analysis is the process of analyzing the code of an application without actually executing the program. On mobile devices, it can be accomplished through various tools, such as *lint* and *Dexter*. Dynamic analysis is the process of analyzing the behavior of the application while running. We refer to offline profiling as profiling the application without offloading, and to online profiling as profiling the application while offloading. In some cases, researchers do not do any automatic analysis at all, but focus on a few applications, for whose behavior they have a priori knowledge.

A.1.2. *Application Metrics* are the metrics extracted during the profiling stage for components of various granularities. They help with the understanding of the application behavior and, thus, are used in the partitioning process and serve as basis for the offload decision. Metrics such as memory usage, CPU time, energy consumption and operational time characterize the behavior of various components from a performance point of view. As the application is split into components of various granularities, the offloading decision can be based on metrics such as the access frequency or the number of invocations of given components, and the size of input/output data of each component, both being used to compute the effort of handling that component remotely. Portability shows if a component can be offloaded or not, as some, e.g. user interfaces, cannot be handled remotely. Location of the component shows if that component has already been offloaded or not, which can have an effect on the offloading decision.

A.2. *Partitioning* serves as starting point in the offload process and aims to divide the application into components organized in a partition or set of partitions. Afterwards, a decision must be made about which components to offload and which components to run locally. This process can vary in complexity from a simple hard-coded decision to comprehensive graph algorithms.

A.2.1. *Partition Mechanism.* When partitioning, different mechanisms can be used. If the application is well-known, then a manual, hard-coded partitioning can be used. However, if the offloading solution should work on multiple applications, more complex methods should be used. The knowledge of the developer can be used by asking him to make code annotations, that is to indicate which pieces of code can be offloaded and which should be run locally. Probably the most complex solution, to employ fully automatic partitioning, is to use the output of the profiling component to represent the application as a graph, in which nodes are components and edges are the communication between components. Each node and each edge can be characterized with specific metrics, and then a number of graph specific algorithms, such as clustering, can be used to obtain an optimal solution.

A.2.2. *Partition Granularity.* Partitioning the application can be performed at various levels. Components can differ in size from an approach to another, ranging from a method or set of instructions, to full threads or processes, and even to entire virtual machines.

B. Resource Management

As shown in Section 2.2.2, the Resource Management is needed to monitor local and remote resources and to supply remote resources. The offloading process is essentially the usage of remote resources. Therefore both aspects are important for having a worthwhile offloading process.

B.1. *Resource Monitoring* assess the resources available on device, such as CPU, memory, battery level and communication capabilities. The specifics of mobile applications can motivate an offload for any of these types of resources, and monitoring itself can be done using several types of metrics.

B.1.1. *Offloaded Resource.* Perhaps the most common form of offloading is to remotely execute code, thus alleviating computation. However, sometimes remote execution is done with a concern for the working memory used by that code, and we refer to this as memory offloading. Communication offloading comes in two flavors: using a proxy, commercially implemented as push notifications services, and using a peer, by sharing file downloads and other communication intensive tasks. Content and storage offloading are also commercially implemented, in the form of Content Delivery Networks and remote storage services.

B.1.2. *Resource Metrics* are used to assess available resources, both locally and remotely. Computation, communication and data storage resources are all covered in existing research. Resource metrics mirror the application metrics presented in A.1.2.

B.2. *Resource Supply* in the environment can be done either through discovery—which is useful in opportunistic approaches, when the device uses whichever resources are available—or through provisioning—a more active approach that can also be found in other types of distributed systems, where the system is able to create new resources in the environment based on the demand.

B.2.1. *Offload Target Placement.* Based on the hierarchy of devices presented in Figure 2.1, the resources can be either in the distant and powerful cloud, the local cloudlet or even among peers. Such a wide range of possibilities arises trade-offs to be studied by the researchers, but also poses some interesting ownership challenges for commercial implementations.

B.2.2. *Discovery Mechanisms.* Often, especially when the research efforts focus on other areas of offloading than resource monitoring, the researchers use statically defined resources, ranging from their own laptops, to virtual machines in the cloud. Others use general distributed computing principles by employing a component that acts as a resource broker, mediating the dialog between the resource hungry device and the resources available in the environment. Finally, a few approaches leverage device collocation when selecting a remote resource, as this usually leads to lower latency times.

B.2.3. *Provisioning Criteria.* If there is any provisioning, it should consider the demand to scale up and down the number of remote resources available for offloading, to optimize performance and cost. Given the nature of mobile applications, provisioning might take into account the specialization of the resources, which might be either the same type as the client mobile device for compatibility, or of a different type for a diversification of the mobile device's capabilities. As an alternative, cyber foraging is an opportunistic approach, that uses no provisioning at all.

C. Offload Process

As shown in Section 2.2.3, the offloading process is iterative, to adapt to fluctuating conditions, and uses information from A. Application Monitoring and B. Resource Management.

C.1. *Offload Decision* gathers some of the most diverse ideas in offloading for mobile devices, depending on the benefit assessment. The approaches differ in the way they assess benefits, how they collect feedback from previous iterations of the offloading process and how they take into account context.

C.1.1. *Benefit Assessment.* Offloading can be targeted for optimizing either performance, energy consumption or costs. The three criteria are usually mutually exclusive, that is minimizing all at the same time is not possible, but a balance between two or more criteria can lead to a good benefit assessment model.

C.1.2. *Feedback Collection.* At the end of an offloading iteration, results computed remotely must be merged with local results, errors must be handled and a general performance assessment must be made.

C.1.3. *Context Awareness.* In many cases, the context can be leveraged for better results. For example, user behavior, social relations between users and location can be used to model and predict workload changes and network conditions.

C.2. *Allocation* refers to the way in which the system decides on what resources to use for which tasks—allocation criteria—and how to use multiple tasks on a limited number of resources—allocation strategy.

C.2.1. *Allocation Criteria.* Various criteria can be found in the research literature for deciding where to offload, that is which of the remote resources to use. They range from performance related considerations, such as allocation time, processing time overhead, energy consumption and network bandwidth, to qualitative resource properties, such as data affinity, and job type.

C.2.2. *Allocation Strategy.* Remote resources can be used exclusively or multiplexed by several clients. The space sharing allocation strategy is done through virtualization.

C.3. *Offload Operation.* The offload operation itself can be met in a variety of conditions, depending on the theoretical mechanism used and the actual implementation. Offloading inherently implies a sort of division between what is done locally and what is done remotely. The division can refer either to data—in this case it can be compared with the SIMD paradigm—or to processing—similar to the MISD paradigm.

C.3.1. *Mechanism* depends on the application partition granularity described in A.2.2. It can range from cloning entire virtual machines, full process migration, job partitioning, to remote code execution.

C.3.2. *Parallelism.* All types of offloading can be done either sequentially or in parallel. Moreover, sometimes a concurrent offloading can be implemented, in which tasks are solved both locally and remotely and the quickest result is used.

C.3.3. *Processing Division.* A full processing can be offloaded, or it can be offloaded partially, either spatial or temporal. For example, in a game that simulates a map over a loop, offloading can be implemented only on part of the map, or only on part of the loops.

C.3.4. *Data Division* is based on the concepts of region of interest and object of interest, that identify the part of the data that benefits from the user attention. For example, in graphics-oriented applications, the viewport is the region that the user sees. Offloading can be implemented for the region of interest, while processing data outside the region of interest can be considered background effort and can be performed locally.

D. Orthogonal Concerns

As shown in Section 2.2.4, several of the topics researchers address in their papers are not part of either Application Monitoring, Resource Management, nor the Offload Process. Therefore, in this area of the taxonomy we identify them as orthogonal concerns.

D.1. *System Properties.* The general properties of the offloading system are a key indicator of the system's functionalities and qualities. However, they refer to the system as a whole and cannot be integrated in any of the three areas of the General Offloading Model.

D.1.1. *Functional Properties.* The distribution property is the geographical spread of the system's components which enables it to perform well regardless of the mobile client's position. Matching hardware is a core operation property that ensures a good functionality for all the heterogeneous devices that benefit from offloading. Lawfulness and privacy are properties that arise from the distributed nature offloading systems have, when the user allows personal information from its own mobile device to travel to infrastructure, which can belong to third-parties, can exhibit multi-tenancy and can use data from sources with different owners.

D.1.2. *Qualitative Properties.* In offloading systems, performance is a key property, defined for each of its components and for the system as a whole. It is often used to settle the expectations of each of the stakeholders, like establishing network latency in a

Service Level Agreement. The availability property refers, like in all distributed systems, to the proportion of time the system is in functioning condition. The reliability property refers to the ability of the system to perform its functions correctly in various conditions, whereas the durability property refers to the ability of the system to perform its functions correctly in lengthy periods of time.

D.2. *Adaptability* can be expressed in many ways and is an important factor for offloading systems, that must cope with various changes in the context. The adaptability of the system matches the ideas from C.1.3. Context Awareness. When offloading is not beneficial, depending on the application, the system can perform quality adaptation. Mobility leads to changes in communication capabilities and offloading systems might support network adaptation. As a last resort, the system can deny new users through admission control.

D.2.1. *Quality Adaptation* is sometimes necessary when the system cannot keep up. For example, a game may choose to reduce its graphics quality, or an image processing application may choose to reduce the number of processed frames.

D.2.2. *Network Adaptation* can be accomplished through handover. A number of protocols, such as IEEE 802.21 and IEEE 802.16e focus on handovers between various network technologies.

D.2.3. *Admission Control* is a classical mechanism in service-oriented systems, which assigns a component for approving or rejecting requests entering the system. Thus, a system that cannot service more requests can simply deny them from the beginning.

D.3. *Security & Privacy.* Whether offloading uses a distant cloud resource or a closer cloudlet resource, data travels from the safe environment of the local device to the outside world. People are already concerned with privacy and security issues on their mobile device, and, in this respect, offloading integrates in the same lines as sharing documents and information.

D.3.1. *Security.* As data and code travels through various networks to reach the remote resources used in offloading, security becomes a basic concern. However, classical mechanisms can be used such as encryption, authentication, authorization and trust management.

D.3.2. *Privacy* is a great concern when offloading. Usually mobile devices are used as personal assistants and people trust them with sensitive personal information, which might be transferred to remote resources, that usually belong to different tenants.

D.4. *Logging*, in a commercial application, is essential for accountability and billing. However, research prototypes also benefit from logging as a functional tool, to help debugging.

D.4.1. *Accountability.* Given the big security and privacy risks that distributed applications in general pose to mobile users and the vast amount of service providers that have a role in the offload process, accountability is a key feature that offloading systems need to have when they become available for the general public.

D.4.2. *Billing.* Offloading implies services and resources supported by many third-party providers. Given the popularity of mobile devices, offloading solutions can also benefit from vast numbers of users, thus becoming an interesting revenue source for the providers.

D.4.3. *Functional Logging.* Offloading is a complex process, with great variations in time due to the changing context. Thus, during the development stages of the offloading system, the researchers use logging as a method for debugging their solution.

2.3.2 Mapping of Existing Research on the Taxonomy

In this section, we provide a comprehensive view on existing literature on offloading for mobile devices, by matching it with our taxonomy.

A. Application Monitoring

Many research efforts on offloading for mobile devices focus only on a couple of applications, thus simplifying the Application Monitoring area. However, there are some which strive for an automated solution, which would work on many types of applications, and thus need to employ complex mechanisms. Some correlations among topics can be made: solutions that employ graph techniques for partitioning also need statistical application metrics, such as access frequency and input/output data size. Moreover, works where the researchers choose applications in which they use a priori knowledge, aided sometimes by static analysis, usually lead to manual partitioning and, thus, a focus on other areas of the offloading problem. Table 2.1 summarizes our findings.

Table 2.1: Mapping of Application Monitoring efforts with our taxonomy.

	Profiling Mechanism	**Application Metrics**	**Partition Mechanism**	**Partition Granularity**
Hassan, 2011 (map-reduce) [33]	D	C,D	S	M
Eom, 2012 (Snarf) [18]	S,D,P	C,P	C	J
Hong, 2009 [19]	S,D	E,T	M	T
Kemp, 2012 (Cuckoo) [36]	S,P	M,C,T	M	M
Huerta, 2010 [22]	S,O	T,L	M,C	M
Lagerspetz, 2011 [31]	S	E	M	C
Balan, 2007 [11]	S	-	C	M
Cuervo, 2010 (MAUI) [23]	D,O,P	P	C	M
Ou, 2007 [15]	S	M,C,P,F	M	C
Gu, 2004 [24]	O	M,P,L,F,D	G	C
Zhang, 2010 [26]	S,P	M,C,E	M,C	C
Satyanarayanan, 2009 [10]	P	-	M	V
Portokalidis, 2010 [16]	P	C	M	P
Verbelen, 2012 [20]	S,D	C,P	M	C
Chun, 2011 (CloneCloud) [3]	S,D,O	C,E,D	G	V,T
Giurgiu, 2009 [21]	S,D	M,P,D	C,G	C
Flores, 2011 [17]	P	C	M	C

Legend: *Profiling Mechanism:* S=static analysis, D=dynamic profiling, O=online profiling, P=programmer input; *Application Metrics:* M=memory usage, C=CPU time, T=operational time, E=energy, P=portability, F=access frequency, L=location, D=input/output data; *Partition Mechanism:* M=manual partition, C=code annotation, S=chunk splitting, G=graph techniques; *Partition Granularity:* V=virtual machine, J=job, P=process, T=thread, C=component/object, M=method

A.1. *Profiling.* Often, when researchers choose one or two applications, profiling is limited to the a priori knowledge the researchers have on them. However, many research efforts strive for automatic, general solutions that employ a mix of profiling mechanisms, with multiple application metrics.

A.1.1. *Profiling Mechanism.* Some of the researchers [10] [16] [17], focus their efforts on one or a few applications, for which the behavior is considered a priori knowledge. Most of the research efforts [18] [19] [20] [3] [21] employ a mix of static and dynamic analysis mechanisms. Notably, there are a few efforts [22] [23] [24] [4] [3] that employ the type of profiling that monitors the application behavior while offloading.

A.1.2. *Application Metrics.* Most of the research efforts to profile applications for offloading measure, per component, the CPU usage and other performance related metrics, such as memory usage, energy consumption and operational time. However, as shown in [21], assessing the CPU cost for mobile devices is very difficult, due to heterogeneity, and can only serve as a rough estimate. Some works [33] [15] [24] [4] [3] [21] additionally employ statistical metrics such as the number of invocations a component has and the amount of data it communicates, which have a role in graph specific algorithms used in the partitioning. Qualitative metrics such as portability and location of the component are also collected for the offloading decision [18] [22] [23] [24] [20] [21].

A.2. *Partitioning* serves as starting point in the offload process. The application is split in components of various granularities and an offload decision is made about each of them. This process is often modeled by representing the application as a graph, where each node is a component and each edge is the communication among two components. Then it is possible to quantify each node and each edge with specific metrics and to obtain a partition or set of partitions by applying clustering algorithms.

A.2.1. *Partition Mechanism.* Many existing solutions conduct offloading on only a few applications and the authors have a priori knowledge about them. A number of algorithms can be used when partitioning, such as clustering. Some solutions rely on code annotation from the developer, while others strive for an automatic approach.

A.2.2. *Partition Granularity.* Partitioning of the application can be performed at various levels. Components can differ in size from an approach to another, ranging from a method or piece of code, to full threads or processes, and even to entire virtual machines.

B. Resource Management

Table 2.2 shows a selection of papers regarding offloading, with a focus on Resource Management, that cover a broad range of approaches. In the extensive literature we studied, it can be noted that opportunistic or cyber foraging approaches are often based on collocation, as mobile devices will use computational power available in their vicinity. It can also be noted that, although most of these works have a focus on resource monitoring, only some of them deal with resource supply, while many researchers prefer to statically define the remote resources in their experiments.

B.1. *Resource Monitoring* is present in most of the works about offloading for mobile devices. All of them select one or a combination of resources to be offloaded and many also employ a mechanism to monitor existing resources on the device and in the environment.

B.1.1. *Offloaded Resource.* Most of the solutions we studied focused on methods to offload computation. However, there are some notable variations. Huerta et al. [22] offload processing that has high memory requirements and thus is not suitable for mobile devices. Hassan [33] and Flores [32] perform offloading to use the data already present in the cloud rather than bringing it to the mobile device. Communication offloading, in the form of push notifications, has already been implemented as a commercial solution by

Table 2.2: Mapping of Resource Management efforts with our taxonomy.

	Resource Metrics	**Offloaded Resource**	**Discovery Mechanisms**	**Provisioning criteria**	**Offload Target**
Ferber, 2012 [28]	T,X	C	S	P	CC
Kemp, 2011 [38]	E,Bat	N,C	S	S	CC, HC
Hassan, 2011 (map-reduce) [33]	C,E,X	C,D	S	S	CC,RC
Huerta, 2010 [22]	$,C,M	C,M	C	F	P
Balan, 2007 [11]	C,M,B,Bat	C	S	F	RC
Yan, 2010 [29]	X,$	C	B	Z	CC
Ou, 2007 [15]	M, C, B	C	S	S	RC, HC, HS
Zhang, 2010 [26]	C, M, Bat, N	C, S, N	C	F	CC, CS
Satyanarayanan, 2009 [10]	C,M,X	C	B	P,Z,T	CL
Flores, 2013 [32]	C,B	C,D	S	S	CC,CS
Marin, 2013 [27]	C,M,B,Bat	C	B	S	CC

Legend: *Resource Metrics:* C=CPU load, M=memory load, N=network latency, B=bandwidth usage, T=total execution time, X=interaction delay (response time), E=energy consumption, Bat=battery level, $=cost; *Offloaded Resource:* C=computation, N=communication, D=data/content, M=memory, S=storage; *Discovery Mechanisms:* S=static, C=collocation, B=broker; *Provisioning criteria:* P=demand, Z=specialization, T=startup overhead, F=opportunistic/cyber foraging, S=predefined servers; *Offload Target Placement:* CC=cloud computing, CS=cloud storage, CD=CDN, CL=cloudlet, RC=residential computers, HC=home computer, HS=home server, HM=home mobile devices, P=peers.

many mobile technology producers, such as Google [37]. Kemp [38] describes this solution as part of an integrated offloading platform called Cuckoo.

B.1.2. *Resource Metrics* mirror the application metrics presented in A.1.2. and are used when deciding whether the mobile device has enough resources to perform the operations locally or not. Most of the approaches monitor some form of computational load [33][22][11][15][26][39][32][27] and battery level on the device [11][26][27]. Many also measure the impact of network transfers as network latency [26], bandwidth usage [11][15][17][27] or interaction delay [28][33][29][39]. Huerta [22] and Yan [29] also take into account the monetary cost of using remote resources.

B.2. *Resource Supply* is a more niche topic, as many offloading approaches are only tested on a statically defined resource pool, on various levels of target placement, such as laptops [11] or virtual machines in commercial infrastructures like Amazon Web Services [28][36][33]. However, some approaches deal with resource supply in the form of discovery or provisioning, where resources can as well be located on various levels of target placement.

B.2.1. *Offload Target Placement.* Kemp et al [38] investigate offloading communication to a single push server. They note that one may want to use elastic cloud resources when load gets too high, but they admit that Cuckoo, their framework, does not support migration of code between servers. Kumar et al [30] highlight the benefits of using the cloud storage, that has beneficial effects on the amount of transferred data and thus on the performance of the offloading process

B.2.2. *Discovery Mechanisms* Zhang et al [26] focus on two scenarios that require massive resources in constrained locations, namely disaster relief and child finding. This type of scenario encourages the use of devices in the vicinity, by leveraging collocation. In [10], the authors describe the Kimberley Control Manager, that supports browsing and publishing services using the Avahi mechanism in Linux. This module acts as a broker in identifying and maintaining connections with the remote resources. In the work presented by Marin et al [27], a component named Cloud Receiver also acts as a broker in deciding which cloud resource to use.

B.2.3. *Provisioning Criteria.* Ferber[28] and Satyanarayanan[10], who use virtualized environments for conducting offloading, measures the demand and provision when necessary, taking into account power consumption and startup overhead. Also notable is the solution proposed by Yan[29] that also considers resource specialization when provisioning. At the same time, the existing solutions having an opportunistic approach [26] do not employ any provisioning at all.

C. Offload Process

Table 2.3 shows efforts regarding the Offload Process in existing scientific literature.

Table 2.3: Mapping of the Offload Process approaches with our taxonomy

	Benefit Assessment	**Feedback Collection**	**Resource allocation criteria**	**Resource allocation strategy**	**Mechanism**	**Parallelism**	**Processing division**	**Data division**
Ferber, 2012 [28]	P,C	N	L	E,S	R,S	S	S	F
Kumar, 2010 [30]	P,E	S	D,EC	E	M,R	S,P	S	O
Hassan, 2011 [33]	P	R,F	D,J,L,N	S	MR	P	S	O
Lagerspetz, 2011 [31]	P,E	R	A, DA, EC	S	R,S	P	F	C
Cuervo, 2010 (MAUI) [23]	P	S	J,A,P	E	R	S	S	D
Yan, 2010 [29]	P,C	R,F	A, EC	S	S	P	F	C
Ou, 2007 [15]	P	R,F	D	S	R	S	S	F
Zhang, 2010 [26]	P,C,E	S,R	A,P,R	S	M	P	S	C
Satyanarayanan, 2009 [10]	P,Q	S	L	S	C	P	F	F
Verbelen, 2012 [20]	Q	R	A,P	S	R	S	S	F
Verbelen, 2012 (AIOLOS) [4]	P	S	J,D,EC	S	M	S	F	F
Flores, 2013 [32]	P	R	P,R	S	MR	P	F	C

Legend: *Benefit Assessment:* P=performance, C=cost, E=energy consumption, Q=quality; *Feedback Collection:* S=state migration, R=result integration, F=handling failure, N=none; *Resource Allocation Criteria:* J=job type, A=allocation time, P=processing overhead, D=data/code affinity, R=runtime, SD=slowdown , EC=energy consumption, L=least occupied resource, N=network bandwidth; *Resource Allocation Strategy:* E=exclusive, S=space sharing, T=time sharing; *Mechanism:* J=job partition, C=cloning/replication, M=migration, R=remote execution, MR=map-reduce, S=Service; *Parallelism:* S=sequential, C=concurrent, P=parallel; *Processing division:* S=spatial, T=temporal, F=full process; *Data division:* O=object of interest, R=region of interest, F=full data, D=deltas, C=chunks.

C.1. *Offload Decision* gathers some of the most diverse ideas in offloading for mobile devices, depending on the benefit assessment. The approaches differ in the way they assess benefits, how they collect feedback from previous iterations of the offloading process and how they take into account context.

C.1.1. *Benefit Assessment.* Most researchers take into account the performance of the mobile device with and without offloading, expressed as running times. Some also optimize the energy consumption, like [30][31][26], or the monetary costs, like [28][29][26]. Satyanarayanan[10] and Verbelen[20] also refer to the quality of the result when offloading, like better image resolution or more accurate matchings. Usually, the researchers adopt a strict, mathematical assessment model, based on equations. However, in [32], the authors present a prototype based on fuzzy logic that takes into account remote information sources. They demonstrate their prototype by uploading videos and applying a map-reduce version of a face detection algorithm.

C.1.2. *Feedback Collection.* Most of the solutions based on remote code execution discuss methods to integrate the results obtained on the remote resources back into the mobile device. Some solutions, based on virtual machines, like [10] and [4] only discuss state integration. Notably, some approaches, like [33][29][15] get into the complexities of handling failures in remote processing. The easiest solution is to acknowledge failures through keep-alive mechanisms and to correct result errors by discarding all the results that come from a faulty remote resource.

C.1.3. *Context Awareness.* In many cases, the context can be leveraged for better results. For example, Marin et al. [27] take into account social relations among users for better understanding the workloads of the applications and Flores et al. [9] build a cloud-based system to gather benefit assessment from multiple clients and base the offload decision on that.

C.2. *Allocation.* Allocation in mobile offloading literature is rarely a topic of focus. Ferber et al. [28] compare several allocation strategies. On the other hand, several solutions leverage cloud resources, like Amazon EC2 [28] and Amazon Mechanical Turk [29] and, thus, rely on the implicit allocation policies of the cloud provider.

C.2.1. *Allocation Criteria.* Ferber et al. [28] investigate allocation time and processing time overhead using remote objects over Amazon EC2. Other researchers adopt various other allocation criteria, like which is the least occupied remote resource [28][40], or where is the data to be processed [30][33][15].

C.2.2. *Allocation Strategy.* Some of the approaches [3][10] use virtual machines in the cloud, thus using a space sharing allocation technique. Others, who work at finer granularities than virtual machines, like component [31] or method [23][4], can also service multiple clients by space sharing them on the same remote resource. Ferber et al. [28] compare exclusive allocation and space sharing allocation based on the least-occupied criteria, and highlight the trade-off they have on various types of tasks. Time sharing can be envisioned as a possibility as well, but is not presented by the works we studied.

C.3. *Offload Operation.* The offload mechanism is usually one of the focal points in most of the papers on mobile offloading, as there are many variations and some correlations with the granularity of the application. Parallel offloading is considered by some researchers, as a method to increase performance, but its benefits are application specific and bounded by the additional complexity that it brings. Processing division is at the core of the offloading process, which usually splits the processing spatially, and, less often, temporally. Data division is rarely considered.

C.3.1. *Mechanism.* Lagerspetz et al [31] focus on file indexing among devices and between devices and the cloud. The offloading process is described as remote execution among devices and as a service from the cloud. Zhang et al [26] propose an application model based on migrating application components named weblets. In [10], the authors use entire VM migration and dynamic VM synthesis to replicate the work. Verbelen et al [20] propose an OSGI-based offloading solution to maximize the quality of an augmented reality application, while still meeting all defined time constraints. In [32], the authors demonstrate their prototype by uploading videos and applying a map-reduce version of a face detection algorithm.

C.3.2. *Parallelism.* In [29], the authors propose a solution that uses the Amazon Mechanical Turk, a service where tens of thousands of people are actively working on simple tasks for monetary rewards, to perform image search. The sollution is parallel, in a sense that each person matches the query to a set of photos. In [26], the authors conduct offloading on an image processing application. A weblet pool is created on the cloud, and images are processed in parallel by pool members. In [10], the authors acknowledge the benefits of parallelism by cloning the VM if the cloudlet is a cluster.

C.3.3. *Processing Division.* A full processing can be offloaded, or it can be offloaded partially, either spatial or temporal. For example, in a game that simulates a map over a loop, offloading can be implemented only on part of the map, or only on part of the loops.

C.3.4. *Data Division.* The processing of the full content can be offloaded, or only that of the region of interest, for example the viewport in graphical applications.

D. Orthogonal Concerns

Table 2.4 shows a selection of papers dealing with Orthogonal Concerns while offloading. It can be noted that very few research efforts address all types of orthogonal concerns, but many address at least one.

Table 2.4: Mapping Offloading Orthogonal Concerns with our taxonomy.

	System properties	**Adaptability**	**Security & Privacy**	**Logging**
Kumar, 2010 [30]	L,P,D	-	S,P	-
Kemp, 2011 [38]	P,R	-	S,P	-
Eom, 2012 (Snarf) [18]	P,H,A	-	S	-
Kemp, 2012 (Cuckoo) [36]	P,H,R	Q	S,P	-
Klein, 2010 [34]	G,P,A	H	-	F
C2DM, 2010 [37]	G,L,P,A	-	S,P	A,B
Wang, 2010 [41]	P,A,R,D	Q	-	-
Hoang, 2012 [35]	P,A	A	-	-

Legend: *System properties:* G=distribution property, L=lawfulness property, P=performance property, H=matching hardware, A=availability property, R=reliability, D=durability property; *Adaptability:* H=Handover /Network Adaptation, Q=Quality Adaptation, -frame skipping, -reducing graphics, -reducing precision, A=Admission Control; *Security & Privacy and Logging:* S=security, P=privacy; *Logging:* A=accountability, B=billing, F=functional.

D.1. *System Properties.* The performance property of the offloading system is discussed by all solutions we studied. Availability, durability and reliability are also discussed quite often [18][34][35][36] as these are key properties when trying to develop an offloading system beyond the prototype phase and into a public solution. In addition, systems such as the ones presented in [34] and [37] also discuss the distribution property of the system, while others, especially the ones using cloud infrastructures, leave that to the cloud service provider.

D.2. *Adaptability.* In [34], the authors describe Heterogeneous Access Management schemes in the context of Mobile Cloud Computing, One of their highlights is the adaptive network access, handover, based on context awareness elements, such as location awareness, network load, user movement predictions. In [41] the authors propose a rendering adaptation technique which can adapt the game rendering parameters to satisfy Cloud Mobile Gaming communication and computation constraints, with a focus on user experience.

D.3. *Security & Privacy.* Kumar et al [30] discuss several privacy concerns related to using cloud resources, giving examples such as bugs, third-party vendors and location tracking. They identify as possible solutions encryption and steganography. Eom et al [18] propose to use SocialVPN, a tunneling technique through Virtual Private Networks that has a double benefit: better security and virtualisation. Kemp acknowledges [36] security and privacy concerns users might have when offloading, but leaves them for future work.

D.4. *Logging.* Given the research nature of the materials we have studied, most approaches are at a prototype phase and only consider logging for functional and debugging purposes[34]. However, it is easy to understand how commercial offloading solutions like Google C2DM [37] and, more recently, GCM, provide logging for accountability and billing purposes.

2.4 Research Directions

Offloading for mobile devices is a hot topic. It has an increased presence in peer-reviewed publications and events in the past few years, quite similar to the growing popularity mobile devices have with the general public.

Improvements in mobile applications derived from offloading are also increasingly visible. For example, the communication offloading technique presented by Kemp [38] is also implemented commercially by major mobile service providers, like in [37]. Also on the market, applications such as Shazam use remote processing as a basic way of functioning. However, many research efforts still face a number of challenges until they can be implemented for the general public. We identify, in the remainder of this section, research directions for mobile offloading which show great potential for future exploration.

Application Monitoring: most of the research focuses on increasingly automatic ways of partitioning applications at an operations level. We believe that application monitoring can also be understood from a workload perspective, with results such as load predictions, that in turn can lead to better resource provisioning and smarter offloading systems.

Resource Management: resource discovery and provisioning is hardly tackled. Many of the approaches refer to opportunistic offloading, resource scavenging, cyber foraging, and so on. However, resource providers, with expertise in discovery and provisioning in cloud environments, are certainly interested in finding out how they can efficiently offer their resources to the mobile software market, a market with hundreds of millions of users.

Offload Process: research can be made on partial and parallel offloading, as well as exploiting the region of interest. Moreover, experimental setup can be better tuned for real-life applications: experiments not with many clients, laptop instead of mobile.

Orthogonal Concerns: adaptation is often a good companion to offloading and sometimes even a better alternative. Finding a good balance between offloading and adaptation can be a novel approach for performance optimization.

2.4.1 Adaptation versus Offloading: a Balanced Choice

Our taxonomy identifies at item D.2. the adaptability of the system as a key orthogonal concern. Adaptation can be performed in terms of quality of the result, network access or admission control. Notably, when offloading is not beneficial, depending on the application, the system can perform quality adaptation. For instance, in a game it might be preferable to lower the graphics quality, or in a video-processing application it might be preferable to process only some of the frames.

Adaptation may be used as a complement or as a replacement for offloading. To assess which technique is more suitable in a wide range of situations, we consider the factors that have most influence on the benefits of offloading:

- *Resource Type:* as described in our taxonomy, item B.1.1., communication, computation and content can all be offloaded. Communication adaptation can be performed in client-server systems as well, by limiting the number of queries the client is doing. Computation adaptation can be performed, for instance, in a game by lowering the graphics quality. Content adaptation can be done by using different types of assets in mobile applications, depending on the connection or screen size. We investigate communication and computation adaptation and offloading in Chapter 4.
- *Application Type:* we have already identified several types of applications that might benefit from adaptation. These are especially oriented on image processing and graphics, but also on transferring numerous messages. On the other hand, applications that might benefit from offloading are usually the ones that already have some type of distributed functionality, that are computationally intensive, or that depend on data available remotely. We investigate applications that follow both criteria in Chapter 4.
- *Process Tuning:* as shown in our taxonomy, item C.3., the process of mobile applications can be tuned for offloading, by exploiting partiality and parallelism. Adaptation inherently exploits process partiality, by allowing parts of an application to run, but parallelism is not suitable. So, in these terms, offloading shows more complexity than adaptation.

In Chapter 4, we conduct an application domain exploration that investigates communication and computation offloading in the context of several applications, thus covering the first two factors. Some elements of the third factor, process tuning, are also covered by the inherent nature of adaptation and offloading. However, to cover multiple aspects of the third factor we define an Exploratory Space that will be used in Chapter 5, in the Experimental Evaluation of our proposed Integrated Offloading System.

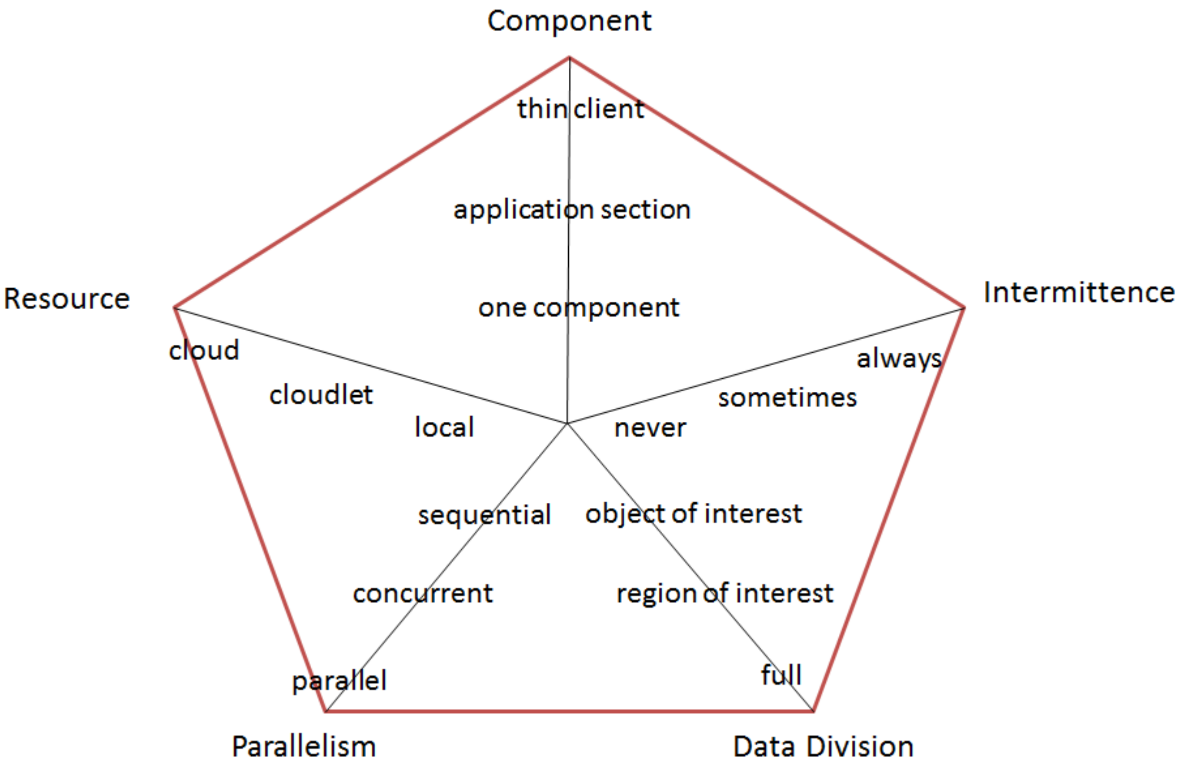


Figure 2.4: Our Exploratory Space for mobile offloading

2.4.2 Exploratory Space

Based on our understanding of current research efforts in offloading for mobile devices, we propose an exploratory space that focuses on topics that show nice research opportunities. Figure 2.4 shows an exploratory space with five dimensions:

- *Component*: using A.2.2. Application Partition Granularity to define application components, the application can be offloaded in various degrees, ranging from a single component to most of its components, leaving only a thin client on the mobile device. As examples of the latter there are various VNC clients for mobile devices and game streaming technologies.
- *Intermittence*: as described by C.3.3. Process Division in our taxonomy, the offloading process can occur partially, either spatially or temporally. Spatial process division is denoted by splitting the application components, which is described in the previous dimension. Temporal process division is described by the intermittence of the process, that is alternation between offloading and not offloading a component.
- *Data Division*: as described by B.1.2., offloading can be performed to alleviate memory limitations. In this case, the data used by computation can be fully offloaded or partially offloaded, either at the level of object of interest or region of interest. This type of mechanism is covered in our taxonomy by C.3.4. Content Division.
- *Parallelism*: as described by C.3.2. Parallelism in our taxonomy, offloading can be done in parallel for some tasks. Also, concurrent offloading could be interesting for studies as a method to maximize performance at the loss of costs.

- *Resource Placement*: as described by B.2.3. Offloaded Target Placement in our taxonomy, using the cloud, the cloudlet or even peers highlights some interesting trade-offs.

Various mixes on the five dimensions can bring interesting results. A comparison of various degrees of offloading components in the context of various available resources has not been made. Also, some of the offloading tuning methods can prove there is still novelty to be discovered in offloading.

To the best of our knowledge, no offloading system has been tested under realistic workloads matching tens of millions of users. Thus, sequentially offloading on remote resources, already more powerful as they are offered by powerful infrastructure such as clouds, has been enough. We plan to investigate parallel offloading as a better method of offloading under workloads such as the most popular mobile applications are showing today. Also, concurrent offloading can be exploited under experimental conditions.

Partial processing offloading and partial data offloading have been studied with limited results. We believe focusing on a specific application domain can bring better insights and a proper understanding of partial offloading. For instance, for games we could investigate the region of interest level of partiality by applying the concept of area of interest in offloading experiments, we could investigate partial spatial offloading by dividing a map and we could investigate partial temporal offloading by processing only some of the frames remotely.

Chapter 3

Workload Modeling for Online Social Applications

We believe that offloading can be beneficial and still pose some interesting challenges for applications which (1) already use some form of communication, and (2) perform most of the functionality by iterating an execution loop. Also considering the popularity various types of applications have, we decide to further investigate Online Social Applications, which are applications dedicated to socially interconnected users, developed for various purposes, such as gaming, multimedia streaming, travel, communication, etc.

3.1 The Context of Workload Modeling

Our work is closely related to studies of Internet workloads, including web applications [42][43][44], peer-to-peer file sharing [45], and online gaming [46]. The research in [44] and [47] brings a focus on access patterns, based on a spectral analysis of data collected during the 1996 Olympic Games. In contrast, our study focuses on online social applications, which represent a new class of applications.

A number of papers deal with characterizing and modeling workloads and performance of web servers. In [43], the authors describe a performance analysis monitor that passively collects packet traces from a server to determine service performance characteristics.

We have also found very useful the efforts to model usage related features in online systems, such as failures in large scale distributed systems [48][49], flashcrowds in bittorrent systems [45] and workloads in online gaming systems [50][51].

Closest to our work, several research efforts focus on online social applications. Nazir et al. [52][53] study the workload of three Facebook applications, with a focus on social interactions. Kirman et al. [54] study two Facebook games in comparison with a stand-alone game, and find that the social networks show a sharp cut-off, in comparison with the scale-free nature of the social network of the stand-alone game. In contrast to this body of work, ours is conducted on a much larger data set and over a much longer period of time, and the focus of our investigation provides new characterization and modeling insights.

Online social applications are currently implemented using various technologies:

- Web 2.0: for example the applications hosted by the Facebook platform and developed by an entire ecosystem of companies; such applications send messages with the user actions to be performed on the server; such applications address hundreds of millions of users only on mobile devices [13]
- Tightly Coupled Simulations: for example multiplayer simulation games, like OpenTTD, which do some heavy processing on the device and usually communicate only control messages; OpenTTD alone has more than 100,000 users on Android
- Streaming: for example online social application and video streaming platforms, like OnLive, which do heavy processing on the server and send video and audio streams to the device; in 2012 OnLive alone was reported to have 1.5 million active users [55]

Applications from all three types of technologies have loop-based processing, with some of the stages occurring remotely, due to their online and social characteristics. Moreover, the social relations between the users lead to some bursty workloads that can prove to be challenging from a research and production point of view. As explained in Section 2.4, we believe that application monitoring can also be understood from a workload perspective, with results such as load predictions, that in turn can lead to better resource provisioning and smarter offloading systems. Thus, in this chapter, we focus on workload modeling on online social applications, and we will use the results presented here in conducting performance evaluation of several offloading techniques, presented in Chapter 5.

3.2 The Data Collection Process

For this study, we have collected several large-scale datasets from online social applications, as depicted in Table 3.1. The objective is to have datasets that cover large time frames and large number of users, and also that come from applications based on different technologies. However, the data collection process is difficult because, generally, application developers are reluctant in sharing usage information for their products. Such information can have negative effects on things like market share or user opinion. Thus, we have quite a diverse collection of datasets, from which only some were fit for the characterization and modeling we present in Section 3.4.

Table 3.1: Dataset overview.

Dataset	Type	total # of applications	total # of users	Duration
facebook-1	users count	630	10M	3 years
	operations	18	2	60 sessions
facebook-2	users count	16,800	10M	3 years
open-ttd	operations	1	10 (AI)	100 sessions
tune-up	users count	500	15,000	9 months
	operations			1000 sessions

The datasets *facebook-1* and *facebook-2* correspond to web-based applications, whereas *open-ttd* and *tune-up* correspond to tightly-coupled simulations. For each of the datasets we describe in the following the data collection process, which is comprised of the crawling, parsing, storing, and sanitizing steps.

We collect two types of datasets: users count or operations. Users count datasets register the number of active users. We use in our work the names given by Facebook to their usage indicators: daily active users (DAU) and monthly active users (MAU). Operations datasets account for the user triggered operations with effects on the computation and communication load. The two types of datasets match the two main sections of our workload model, the Macro Model and the Micro Model respectively, as presented in Section 3.3.

3.2.1 Web-based applications

To collect data for web-based applications, we gathered usage information and traces for Facebook applications using third-party platforms and the Facebook OpenGraph API. Facebook reports daily, through its API, usage data concerning applications dedicated to its users. A number of third-party websites collect this data and report it, over a period of time, in ranks or individually. In our analysis we use DAU and MAU to investigate both popularity and the evolution of the number of users. However, we are also interested in other data that might provide insights, such as application developer, category and subcategory. We use *app* as an abbreviation for application.

Crawling directly application pages on *facebook.com* is impractical due to the changing login mechanism, and the varying page structure and layout. Another alternative is to crawl the Facebook OpenGraph API at *graph.facebook.com*, which retrieves a lot of useful information in easy-to-parse JSON format, but requires the Facebook app id to be known in advance. Thus, we first obtained the *facebook-1* dataset from the third-party websites, and then used the app id list to obtain the *facebook-2* dataset from the Facebook OpenGraph API.

The facebook-1 dataset: user count

In the first stage, we *crawled* two websites, namely *appdata.com* and *developeranalytics.com*, daily for thirty-one months, between January 1st, 2010 and July 31st, 2012. We chose these websites because they report extensive data about a large number of Facebook applications, in a paged-tabular form, covering 40 and 50 applications on each page, respectively. From such data it is easy to obtain an index of applications, sorted by the number of users. Moreover, *developeranalytics.com* displays charts with historical data, going back to the beginning of 2008, which were used to obtain an increased coverage for 50 applications. From *appdata.com* we started to extract MAU and rank for the top applications fitting on 150 pages, for a total of 6,000 applications. However, in March 2010, an unforseeen change in the query parameters lead our crawlers to retrieve in each subsequent session the first page for 150 times. So, in the end, we have information about top 6,000 applications for three months and top 40 applications for thirty-one months. From *developeranalytics.com* we extract DAU, MAU, rank and developer for the applications in top 100, over the same thirty-one months. Also, for a sample of 50 applications,

Table 3.2: Volume of crawled, parsed, and stored information for Facebook applications.

Source	Files	applications	Samples
appdata.com	47,056	16,664	1,864,812
developeranalytics.com	49,177	630	133,594
graph.facebook.com	16,901	16,901	16,901

that were in the top during our first day of crawling, we gathered information for the whole period.

At the end of the crawling period, we developed Python parsers that we used to *parse* and *store* useful data in an unified SQLite database. We completed the datasets with information regarding developer, category and subcategory by crawling *graph.facebook.com* for all the applications obtained from the other two sources.

To *sanitize* the data, we tried to identify and remove any reporting and reading errors. For example, some applications were reported to have an exact same number of users for a few days in a row, or to suddenly have no users for sporadic dates. While this can happen in real-life, due to special events such as failures in the infrastructure, we consider that such features are most probably reporting errors, especially when they affect the whole population of applications. Plotting the evolution of DAU for a few applications shows that the number of users can display a bursty behavior, as it will be shown in subsection 4.2.1. We cannot presume that this behavior is caused by reporting errors and, thus, we investigate this situation.

A quantification of the volume of crawled, parsed, and stored data can be found in Table 3.2.

The facebook-1 dataset: operations

To *crawl* data regarding operations users perform in web-based applications, we used two popular network sniffers, namely Fiddler and Wireshark, while performing 60 sessions on our own devices, with 18 Facebook applications. The network sniffers intercept and store all network packets, with specified filters. Fiddler in particular only intercepts HTTP traffic, which is reasonable to assume consists most of the traffic for the web-based applications.

To *parse* the data we use Python libraries that are able to read the *pcap* trace files produced by Wireshark. The *saz* trace files produced by Fiddler are essential zip archives, each containing a directory structure where all the registered packets can be found as files. We use Linux tools, like grep and awk, to parse the packet files and extract useful information from the headers and bodies of the HTTP messages.

Although we took special care that the browser window with the Facebook app was the only user application using the Internet, we need to acknowledge that other network traffic sources may exist, triggered by the background processes or the operating system. We consider that the Fiddler traces may be less biased, as they contain only HTTP traffic. To further *sanitize* the data, we plotted some of its statistical characteristics. This way we found out that many of the packet sizes were consistent with the standard Ethernet frame size, with a payload of 1500 bytes. However, some exceeded this size. To the best of our knowledge, the network cards we used do not support jumbo frames, so there might be some inconsistency in reporting from Fiddler. In any case, we created a script to repack all frames and to obtain the actual packet sizes.

We do not perform any additional *storage* of the data, because the traces are already in a clear standard file structure and the scripts we used for parsing are quick and versatile.

We also acknowledge that the number of users for this dataset is a serious limitation. We are working to repeat the data collection process using more student volunteers. However, we believe that online social applications based on web technologies have to some extent a number of basic common characteristics at an operations level, which we try to identify in Section 3.3.4 and we use this dataset that, due to the large number of sessions and applications, has statistical value.

The facebook-2 dataset: user count

To *crawl* the data from the Facebook OpenGraph API, we had to create a Facebook user and a Facebook application of our own. This allowed us to obtain a security token that is used in all the queries on the API. The purpose of the token is to allow Facebook applications to require information about other applications on the platform, for a time frame of up to three years in the past. While it seems reasonable to allow all the applications of a developer to require such information one from another, it was surprising for us to see that the API supports queries about any application, from any developer, if the app id is known in advance. Thus, we implemented crawling scripts in Python that run for two days on a machine in our cluster, downloading the full history for the whole list of application ids in the *facebook-1* dataset.

Parsing the data was easy because of its standard JSON format, each response from the server containing, besides information about DAU and MAU, the application title, developer, genre, category and link. We *store* the data in the same unified SQLite database as the *facebook-1* dataset.

To *sanitize* our data, we compared the two datasets. Mostly, the results matched, but some timestamps with null values in the facebook-1 dataset did have values in the facebook-2 dataset, indicating that some information was lost because of the indirection to third-party websites.

3.2.2 Tightly coupled simulations

Collecting data for tightly coupled simulations is not an easy task. Even when the applications perform communication with a public server, a significant amount of the workload takes place on personal devices. People are usually reluctant to offer any kind of usage information regarding their devices. On one hand, we collected the open-ttd trace by using OpenTTD ourselves. Though the number of users is limited, we believe there is yet statistically relevant information that can be extracted from this dataset. On the other hand, we made a partnership with Bitdefender, a popular software company that people trust for their security and system tuning applications. In particular, they develop

Power Tune-Up, an application that can be installed on Android devices and collects traces regarding system usage to make some recommendations that prolong battery life and increase system performance. We obtained access, within the boundaries of an NDA, to some of the statistical information that Tune-Up collects. We describe our effort to collect data from Tune-Up in [56], but we do not further detail it here, as its results are not used yet in the workload model.

The open-ttd dataset: operations

For covering this type of applications we have conducted profiling on OpenTTD, a popular open-source simulation game. In multiplayer mode, OpenTTD currently supports up to 255 simultaneous users on the same map, one of which must host the server. Each of the clients needs to perform every 100ms a processing loop, consisting of input capture, AI input (if AI's are used), command synchronization, simulation and render. During command synchronization, each client sends all the commands issued by human or AI players, the server puts them in order and sends the whole collection of commands back to the clients for execution. We further detail the functionality of OpenTTD in Section 4.2.2.

To *collect* run-time data from OpenTTD, we modify the source-code and log timestamps at key moments in the processing loop, using the *time* function in C++, which has a precision of the order of microseconds. We choose the key moments by profiling the application. We use *gprof* to obtain the call graph for the top-level functions, handling the six stages in the loop. We *process* the data by computing the first derivative, to obtain the time spent on computation in each stage. It is important to note that the commands issued in one iteration of the loop are transmitted to the server, received back by the client and processed in the next iteration. Thus, the time spent on command synchronization will be more than the time interval between two loops (that is 100ms), but the total time spent on such a loop should not exceed twice the time interval (that is 200ms), otherwise the client may be kicked out of the game, as being too slow. To *sanitize* the data, we identify and remove any outliers, that differ from the median with more than twice the standard deviation.

To *collect* network traffic, we use *tcpdump* while running the multiplayer game and output the trace to a text file. We *parse* the text file with *awk* and *store* the information in csv files.

These collection procedures serve as a base for the test-bed we design and implement using OpenTTD, which we describe in Section 5.2.

We use a Sony Vaio 4-core @2.3GHz running Ubuntu 10.04 (*laptop*) as server, and a Asus TF101 2-core @1GHz running Android 4.0.3 (*tablet*) as client. Given the diversity of parameters that govern the game, we choose a standard Scenario (*Europe*) which benefits of a large map of 1024x1024 tiles, numbering tens of thousands of cities and resources, and we set a high graphic detail and animations. We further discuss how various parameters can influence performance in Section 5.2.

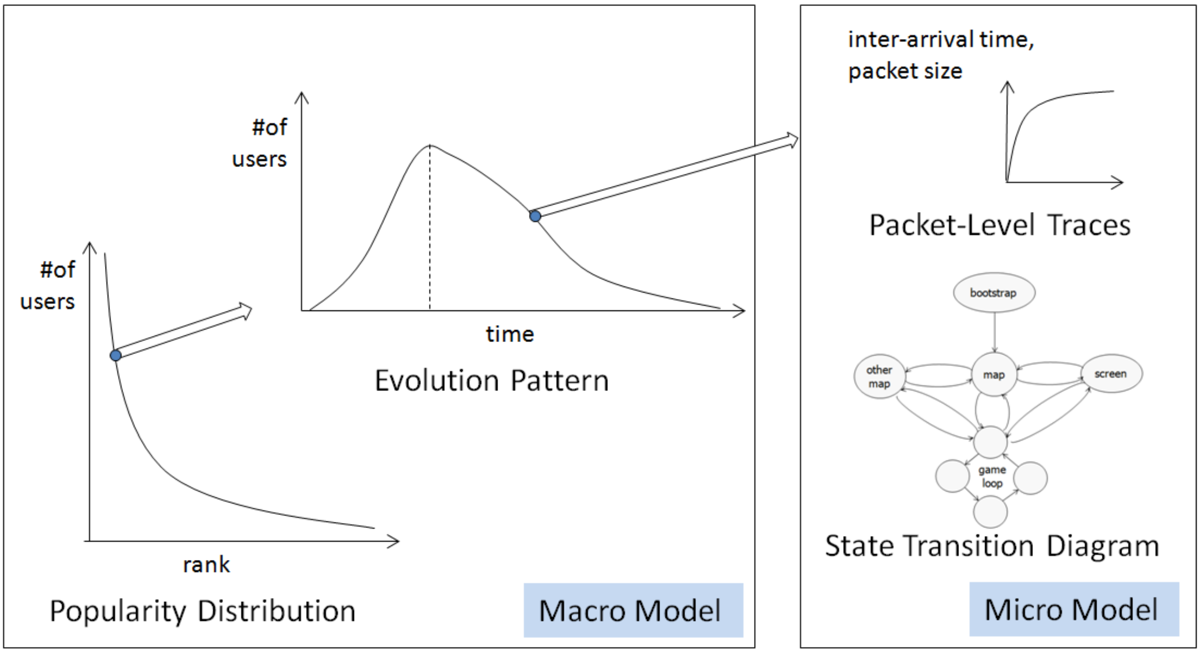


Figure 3.1: The Workload Model

3.3 The Workload Model

To help understand the loads on online social applications we create a workload evolution model (see Figure 3.1). The key element of our model is evolution, either short-term (time range spanning the lifetime of a online social application session, up to a few hours, with a resolution of milliseconds) or longterm (time range spanning the lifetime of a online social application, so up to several years, with a resolution of one day). To accommodate the two very different time ranges and accuracies, we propose separate model components:

- a *macro-evolution* component for the long-time range, which accounts for the number of active users applications have, consisting of three elements: the popularity distribution—which describes how users are spread throughout a population of applications—the evolution pattern—which describes how the number of users varies during the lifetime of an application—and the daily pattern—which describes how the number of users varies during a day
- a *micro-evolution* component for the short-time range, which accounts for the operations users trigger and their associated load, consisting of two elements: the general network statistics—with insights into network traffic and processing times—and the state transition diagram—which segments loads on different application states.

We detail our model in the remainder of this section, and present characterization and modeling results in Section 3.4, using the datasets described in Section 3.2.

3.3.1 Popularity Distribution

To quantify application popularity within a population of similar applications, we investigate the relation between maximum number of DAU and the rank of the application. We expect this relation to conform with a Pareto-like principle, with top ranking applications gathering many more users even than the ones ranked immediately below them.

We also expect that the popularity distribution is not time-varying, despite the overall increasing number of users mobiles and online social applications have. We also expect that the popularity distribution is not affected by seasonality, given the global scale of the user base. To validate this expectation, we use our long-term datasets and present the results in Section 3.2.

To model popularity, we conduct curve fitting on the empirical distribution for various reference distributions, namely Exponential, Weibull, Pareto, Log-normal, and Gamma. We use the maximum likelihood estimation method (MLE) to estimate the distribution parameters, and the Kolmogorov-Smirnov (KS) test to evaluate goodness of fit (GOF). The results are reported in terms of the p-values and D-values, as well as the parameters that provide the best fit for each of the reference distributions. The p-value shown is the average of 1000 values, each of which was computed by selecting 30 samples randomly from each dataset.

3.3.2 Evolution Pattern

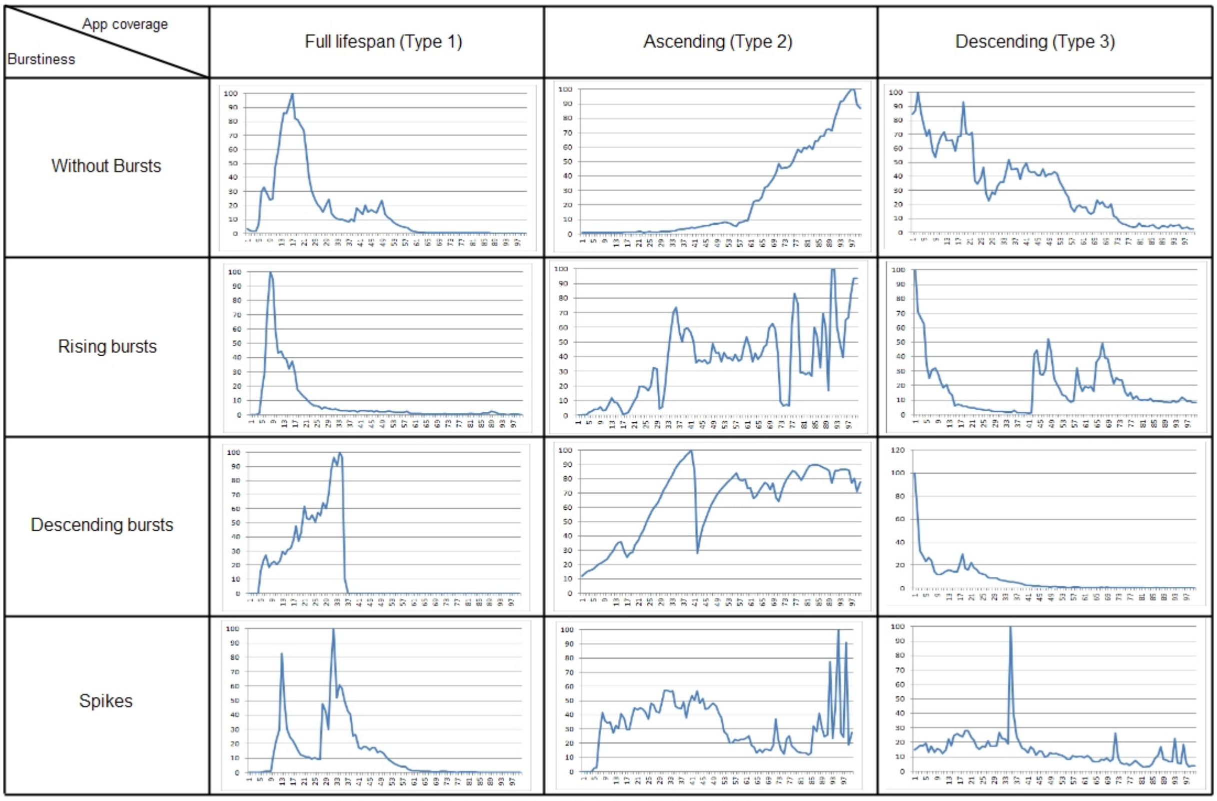
Networked applications can evolve in various ways over time. An infamous evolution typology is the flashcrowd (burst, spike) [45], for which the number of users over time first grows quickly, then reaches a peak high above the number of users at the start of the flashcrowd, then quickly decreases. When subjected to this type of workload, many services crash or severely depreciate their quality of service. However, other evolution typologies also exist.

In this work we identify several major evolution typologies, each of which is governed by parameterized evolution patterns. We develop a model for each pattern and an algorithmic classifier that maps a given OSG to its evolution typology. We validate our model in Section 3.4.

Orthogonal to the evolution typologies, but of importance to the algorithmic classifier, we also identify *three* application typologies to represent application coverage—whether the observation data captures the full lifespan of the application (*Type 1* of coverage), an ascending part (*Type 2*), or a descending part (*Type 3*).

We identify the following *five major evolution typologies*: stationary, peaked but without bursts (usually single-peaked), with rising bursts, with descending bursts, and multi-peaked (with spikes), which we depict in Figure 2 and for which we propose models:

- Class-1 (*stationary*) applications have a relatively constant DAU, with DAU values not varying by more than 10% from the mean computed for the first 10% of the application's lifetime.
- Class-2 (*single peak*) applications grow steadily to a peak, then decay steadily (Figure 3.2, row "without bursts"). The growth and decay occur slowly, and may be interrupted by limited periods (maximum 2 weeks) of opposite effect. To model this typology, we use a linear model consisting of two line segments. The parameters of the evolution pattern for this class are the peak value, the growth rate, and the



Horizontal axis: % of apps lifetime | Vertical axis % of peak DAU

Figure 3.2: Classes and typologies of evolution

decay rate. The peak value (p) occurs at t_p and is measured as the maximum DAU throughout the lifespan of the OSG (t_l). The growth rate is $m = \frac{p}{t_p}$ and the decay rate is $n = \frac{-p}{t_l - t_p}$. The relation between growth rate and decay rate can be quantified by the maturity of the OSG at peak $r = \frac{t_p}{t_l}$. Growth and decay are related: $m = (1 - \frac{1}{r})n$. For each of the four parameters described in this subsection, we plot the cumulative distribution function (CDF), and we conduct curve fitting following the same procedure that we use for fitting the popularity distribution.

- Class-3 and Class-4 (ascending and descending *bursty pattern*, respectively) applications resemble Class-2 games, but exhibit *sharply* ascending or descending periods (Figure 3.2, rows of “rising” and “descending bursts”). Identifying this pattern may be difficult, for example because bursts may oscillate over short periods during their lifetime, leading to false positives. As in our previous work on flashcrowds occurring in peer-to-peer systems [45], we model the bursty pattern using the start time (measured after the birth of the OSG), the duration, the duration of the peak period (the plateau), and the magnitude of the major flashcrowd (measured as a ratio between the peak number of users and the number of users at the start of the flashcrowd).

- Class-5 (*multi-peaked*) applications (Figure 3.2, row “spikes”) combine the properties of multiple Class-3 and Class-4 applications, and possibly also exhibit periods during which they behave as Class-1 or Class-2 applications. Thus, they are difficult to model and predict.

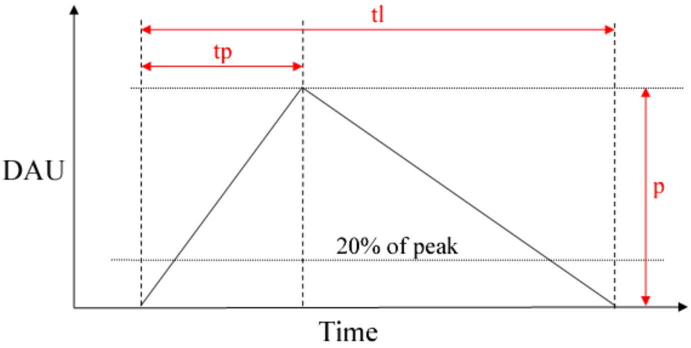


Figure 3.3: A simplified model to approximate the growth and decay of the number of users for social applications. Labels: p, peak value; t_p, time to peak; t_l, total length of life.

3.3.3 Packet-Level Traces

The Packet-Level Traces element models the network load that online social applications have, in terms of packet sizes and inter-arrival rate.

Online social applications can transfer images, multimedia, data and actions. Being online, these applications are already optimized for a low latency and often use images in compressed formats, like jpg and png instead of bmp, multimedia packed as swf files, data represented as JSON strings and actions as scripts in various languages. However, they can still transfer over the network tens and hundreds of megabytes in one session, which makes the quantity of transferred data interesting to model. The messages however can vary in size with orders of magnitude, from control messages without any payload or just a few bytes to content messages of megabytes of data. Small messages however can lead to a considerable network load as, regardless of how small the payload is, each packet contains headers from various levels in the communication stack. Big messages as well need to be split into sizes that fit into data frames, each message will gain in size with several headers.

The inter-arrival rate complements the packet-size in understanding the network load, as it also has a great influence on the resource consumption. Wireless communication modules consume a lot of energy when in active modes. Once a packet is sent, the module will remain active for some time, consuming energy without actually transferring any useful data. Therefore, regardless of the size, small and frequent packages can lead to even a greater power consumption than a few large packages.

Thus, to better understand the real impact an application has on the network, we compute the cumulative distribution function (CDF) for packet sizes and inter-arrival times. The CDF depicts the full range of values each of the two metrics can have, also showing which are encountered more often, and can be complemented with a time-based profile for better understanding of how the two metrics vary in time.

Packet level information can be segmented on multiple criteria. For example, an application developed by Zynga and hosted by Facebook normally makes queries to 10-20 servers, belonging to Zynga, Facebook or other companies, such as Google that offers analytics services. Also, the servers are usually dedicated to various purposes: authentication,

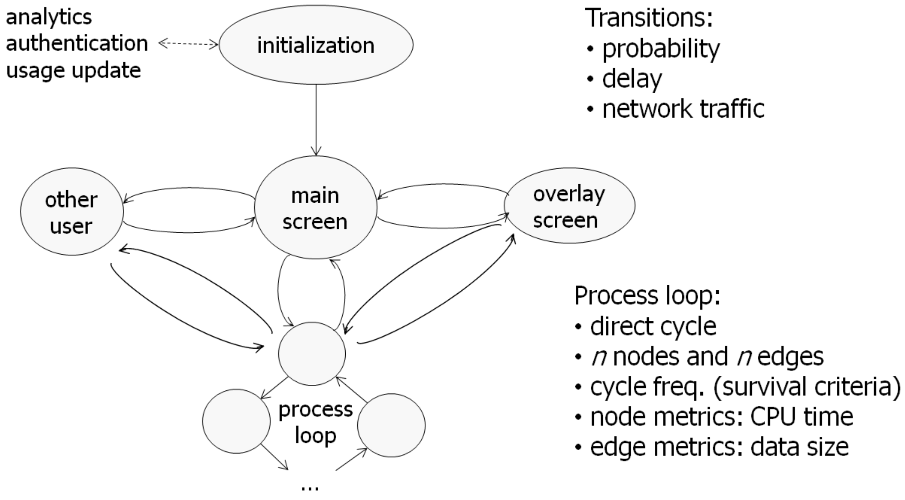


Figure 3.4: The State Transition Diagram

content delivery, command processing, data storage, etc.

3.3.4 State Transition Diagram

We find that online social applications seen as state machines, regardless of the implementation technology, are similiar. Both web-based applications and tightly coupled simulations behave very much in the same way. First, there is an initialization state, in which the application authenticates the user and downloads any necessary assets. Then, the application moves to a principal state, when the user sees a map or otherwise a main screen. While here, when triggered by the user or at fixed time intervals, the application iterates a processing loop. Besides interacting with the application while in the processing loop, the user can also open a secondary screen, like a settings screen, or activate a social component, like write in a chat window or visit a friend's map. While in these states, the application usually keeps iterating the same loop.

Inspired by the research on Application Partitioning, as described in our Taxonomy (see Section 2.3) we propose modeling the states in which an online social application can be. Figure 3.4 represents these states as a graph we name the State Transition Diagram.

Each node in the State Transition Diagram represents a processing stage and can be described by the time and the resources spent in that stage. Transitions can occur either with a certain probability, or at fixed time intervals. Moreover, the transitions can also model the data dependencies between stages, which can be represented with the quantity of data that needs to be transferred, through the network if the stages do not share memory or virtually if the stages share memory.

While in the initialization state, many applications download most of the assets they need while initializing, to prevent any delays further in their functionality. Thus, we are interested to understand this state first of all from the point of view of network load. Also, the application spends most of it time iterating over the processing loop, so we will try to understand it from a general performance point of view. In our work, we refer to loop-based applications as applications that perform such a loop and can be modeled

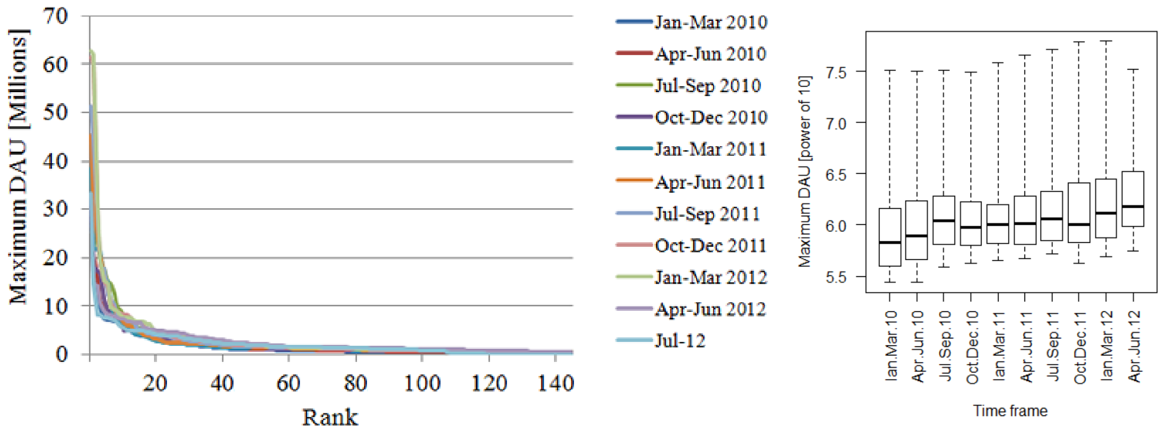


Figure 3.5: Popularity distribution, depicted as the distribution of maximum DAU over rank. One curve per trimester.

with this graph.

3.4 Characterization and Modeling using Real-World Traces

In this section we present empirical results related to our workload model. We characterize and model each element using the datasets introduced in section 3.2.

3.4.1 Popularity Distribution

We analyse the maximum observed DAU and MAU as metrics of app popularity. To account for seasonal transitions yet still be able to follow the results, we analyse the information we have for these metrics per trimester. We characterize and model, in the remainder of this subsection, the eleven trimestrial datasets resulting from our 31 months of data.

Characterization

For each trimester in our datasets, we select the maximum DAU for each of the applications, then sort the maxima in descending order to obtain a rank. In Figure 3.5, we plot the distributions (left side) and a box-and-whiskers chart that offers a more precise view of the main statistical characteristics of the distributions (right side).

Each trimestrial curve exhibits the same phenomenon: **the best-ranked 5-10% applications** (applications rank 1 through 11-21) **attract many more users than all the remaining applications taken together**. Specifically, the first 20 applications have each over 5,000,000 DAUs; many of them also have over 20,000,000 MAUs (not depicted here). This phenomenon and the observed values indicate that there may be two types of app operators: operators of the first 20 applications, who due to their application sizes could largely own the physical infrastructure on which their applications operate, and the other operators, who could lease infrastructure from a cloud, only when needed.

We are also interested in exploring seasonality in this dataset. The shape of the curves for different trimesters are very similar, which indicates that seasonal effects do not influence the relative order of maxima (Figure 3.5). The increasing trend in the total number of Facebook users lead, at the end of 2011, to an accentuation of the differences among the top applications in terms of maximum DAU, which can be seen on the right-hand chart as the box and whiskers are larger.

Modeling through Curve Fitting

We conduct curve fitting for all the eleven trimesters, using the Kolmogorov-Smirnov test (K-S test). The Kolmogorov-Smirnov statistic quantifies a distance between the empirical distribution function of a sample and the cumulative distribution function of a reference distribution. We choose five reference distributions, Exponential, Weibull, Pareto, Log-Normal and Gamma, which cover the most cases we have found in literature. For each reference distribution, the K-S test provides the parameters that provide best fit, the p-values which show the level of significance of the result and the D-values which quantify the distance between our empirical distribution and the best fit.

We present the p-values and the D-values in Table 3.3. We dismiss the distributions for which we obtain a p-value below the significance level of 0.5. The D-values show that **the Log-normal distribution provides the best fit with the data we analysed, from the five distributions we have tried**, except for the last timeframe, which actually consists of a single month. The parameters obtained for each distribution are summarized in Table 3.4.

Table 3.3: p-values and D-values resulting from KS tests conducted for the popularity distribution over five reference distributions: Exponential, Weibull, Pareto, Log-normal and Gamma.

series	Exponential		Weibull		Pareto		LogNormal		Gamma	
	p-val	d-val	p-val	d-val	p-val	d-val	p-val	d-val	p-val	d-val
Jan-Mar'10	0.073	0.23	0.064	0.22	0	0.65	0.238	0.13	0.079	0.21
Apr-Jun'10	0.083	0.23	0.092	0.19	0	0.67	0.284	0.13	0.099	0.21
Jul-Sep'10	0.056	0.24	0.063	0.2	0	0.66	0.282	0.12	0.066	0.22
Oct-Dec'10	0.065	0.23	0.044	0.2	0	0.69	0.202	0.15	0.066	0.23
Jan-Mar'11	0.038	0.27	0.027	0.24	0	0.7	0.218	0.16	0.039	0.25
Apr-Jun'11	0.049	0.24	0.027	0.25	0	0.66	0.228	0.14	0.048	0.22
Jul-Sep'11	0.05	0.26	0.031	0.24	0	0.69	0.223	0.14	0.049	0.23
Oct-Dec'11	0.055	0.27	0.045	0.22	0	0.65	0.24	0.15	0.063	0.22
Jan-Mar'12	0.057	0.27	0.051	0.22	0	0.68	0.233	0.17	0.065	0.23
Apr-Jun'12	0.132	0.19	0.168	0.16	0	0.69	0.32	0.13	0.196	0.18
Jul'12	0.205	0.15	0.125	0.2	0	0.59	0.044	0.28	0.155	0.2

Table 3.4: Parameters for the reference distributions (Exponential, Weibull, Log-normal and Gamma) that provide best fit for the popularity distribution. Values for μ_e, λ and θ are expressed in multiples of 10^6

Series	Exp(μ_e)	Wbl(k_w, λ)	LogN(μ_l, σ)	Gam(k_g, θ)
Jan-Mar'10	1.76	0.82 1.53	13.69 1.03	0.84 2.09
Apr-Jun'10	2.03	0.84 1.79	13.86 1.01	0.87 2.34
Jul-Sep'10	2.64	0.83 2.29	14.10 0.99	0.86 3.09
Oct-Dec'10	2.13	0.89 1.96	13.99 0.90	1.00 2.14
Jan-Mar'11	2.31	0.84 2.03	14.02 0.90	0.92 2.52
Apr-Jun'11	2.55	0.82 2.18	14.08 0.93	0.87 2.93
Jul-Sep'11	2.94	0.82 2.50	14.20 0.95	0.85 3.47
Oct-Dec'11	3.22	0.77 2.58	14.20 1.02	0.76 4.24
Jan-Mar'12	3.43	0.80 2.84	14.31 0.99	0.81 4.26
Apr-Jun'12	2.73	1.06 2.82	14.41 0.83	1.36 2.01
Jul'12	2.17	0.63 1.65	13.27 2.67	0.48 4.51

3.4.2 Evolution Pattern

We analyze in this subsection the evolution of the values observed for app DAU and MAU. Similarly to the previous subsection, we first present characterization, and then modeling results.

Characterization

To compare applications while accounting for the high variability among them—both lifespan, and evolution of DAU and MAU—, we first normalize both the lifespan and the number of users of each application. In the characterization plots presented in this subsection, we express for each application the number of users at a moment in time as a percentage of the application's peak number of users, and the moment in time as a percentage of the application's total lifetime.

In our datasets, we have at least one DAU reading for 16,799 applications. To make better use of our data, for evolution characterization we select a sample of applications. First, we disregarded applications with very few readings, i.e., less then 100, for which normalization would actually reduce resolution. We further select only those applications that reach a peak of at least 5,000,000 DAUs or 20,000,000 MAUs. These are thresholds inspired by the results obtained while characterizing application popularity (subsection 3.4.1); these applications account together for more than 80% of the total DAUs and MAUs observed in our datasets. On this sample we performed manual classification, going through tens of charts like the ones presented in Figure 3.2, depicts the DAU evolution for several applications in the sample, with normalized values.

We then extend our sample to all the applications that reach a peak of at least 100 DAU. Although this sample limits the number of applications to 5,723, we consider, based on the results obtained while characterizing application popularity (subsection 3.4.1), that all other applications do not posses a significant impact on the general user population. Classifying thousands of applications manually is, of course, unfeasible. This led us to

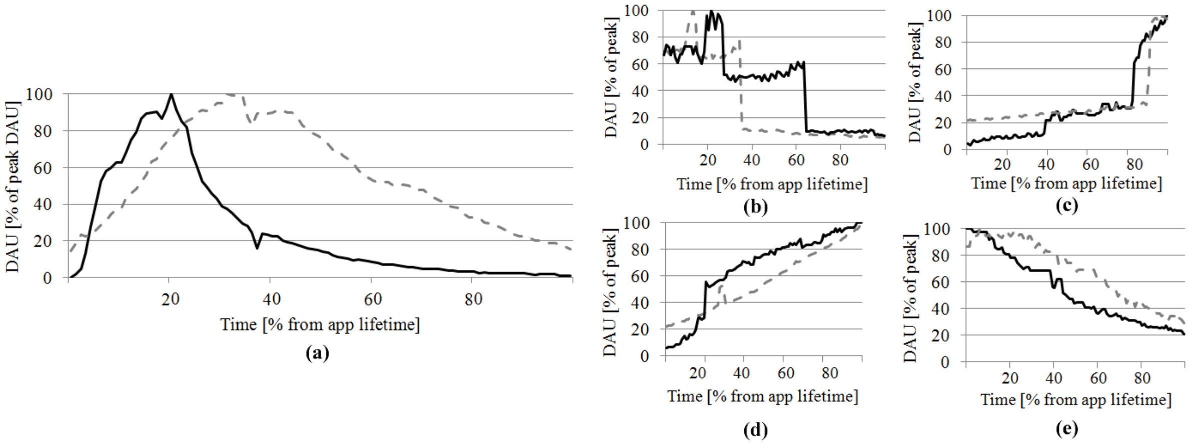


Figure 3.6: Examples of DAU evolution for several applications: (a) typical single-peaked pattern, (b) suddenly ascending, (c) suddenly descending., (d) continuously ascending, (e) continuously descending.

develop an algorithmic classifier. The design and implementation of the algorithmic classifier were done with the help of Emanuel Dias, from Delft University of Technology. To account burstiness, the classifier accounts for large values in the first derivative of the time series. To account for the lifespan, the classifier compares the initial value, the final value and a band of values in the middle of the time frame. Table 3.5 shows the results we obtained, expressed as the numbers of application in each of the 5 classes.

Table 3.5: Classification results showing how many applications fit each typology (left) and class (right)

	Full Lifespans	Ascending	Descending
Without bursts	460	32	871
Rising bursts	73	127	198
Descending bursts	66	51	1075
Spikes	596	240	1004
% total	25%	9.50%	66.70%
Constant: 319 (5.57%)	Other behaviours : 611 (19.7%)		

	Count	Percent
Class-1	319	5.57%
Class-2	1363	23.82%
Class-3	398	6.95%
Class-4	1192	20.83%
Class-5	2451	42.83%

The single-peak pattern occurs in less than 25% of all cases, which leaves a majority of applications exhibiting un-common patterns and, thus, are difficult cases for provisioning.

We are also interested to determine any seasonal and cyclical components in the evolution of the number of users. Inspired by several works on statistics [57], we study the autocorrelation of the DAU as a function over time, following three steps:

1. remove the trend component by differentiating the function

2. calculate a correlogram (a depiction of the autocorrelation function) with a maximum lag equal to the length of the observations vector

3. test values outside the error band $\pm\frac{2}{\sqrt{N}}$, with N number of observations or use the Ljung-Box test to determine any significant outliers on the correlogram

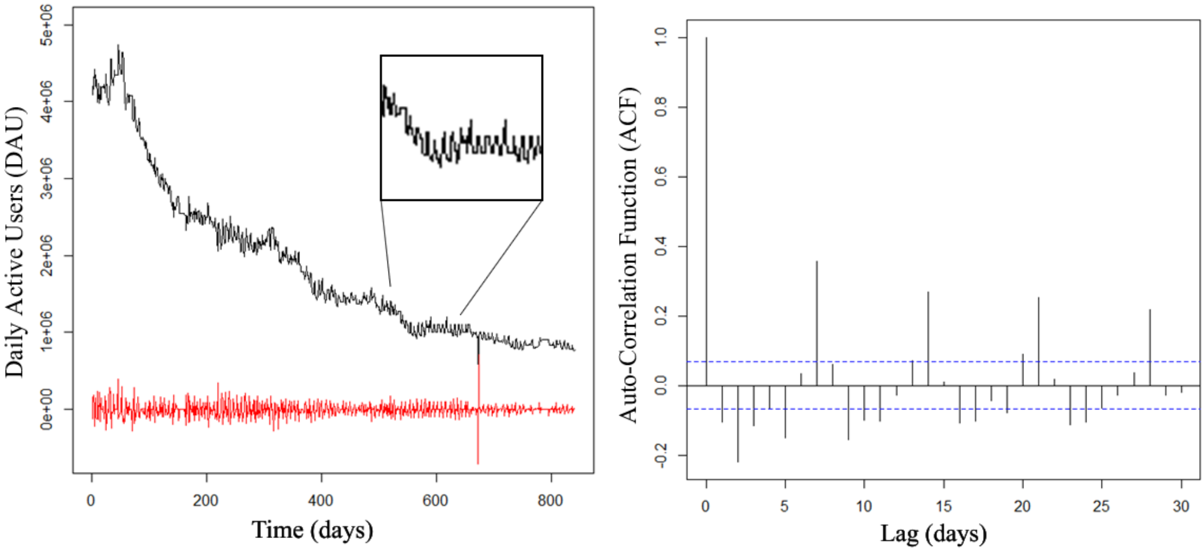


Figure 3.7: Correlogram (right) that shows how the seasonality detection algorithm works on a typical application (left)

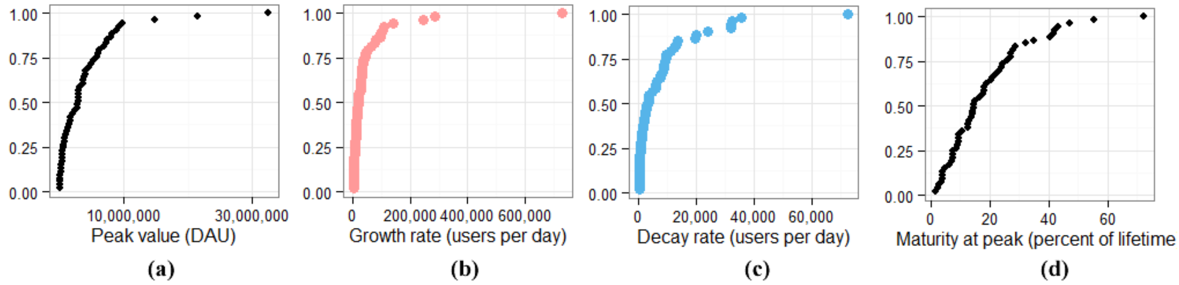


Figure 3.8: CDFs for the parameters of the evolution model: (a) peak value, (b) growth rate, (c) maturity at the peak and (d) decay rate.

Figure 3.7 shows how the seasonality detection algorithm works on a typical application. It can be noted how the autocorrelation function raises above the threshold at values multiple of 7, indicating a weekly seasonal pattern. The most applications we found, especially games, present such a pattern.

Modeling through Linear Fitting

We validate the linear model proposed in subsection 3.3.2 empirically, using our datasets. However, with the current version of our linear model, we can only use data coming from applications for which we have a full lifespan coverage. Moreover, the linear model does not fit well the bursty behavior. Thus, in this section, we can use the 460 applications that fall in Class-2 and for which we have the full lifespan in our dataset.

Using data from the selected applications, we plot a cumulative distribution function (CDF) for each of the parameters of the evolution model, as depicted in Figure 3.8. We find that the values of growth rate and decay rate differ with one order of magnitude (Figure 3.8b,c). Thus we could presume that usually the peak comes rather early in the life of an application: after a quick growth, follows a slow decay. This fits with some of the applications included in the characterization (Figure 3.2a), but not with all of them. To better understand this relation, we compute Pearson correlation coefficients (r) for peak value (p), growth rate (m), and decay rate (n). We find that $r(p, m) = 0.57$, $r(p, n) = 0.30$ and $r(m, n) = 0.41$, which shows there is quite often a high peak in the number of users leads to a steep decay.

A CDF of maturity at the peak is given in Figure 3.8d: 52% of the applications have a peak within 15% of their lifetime and 85% have a peak within their first third of their lifetime.

To model the distributions for each of the parameters, we conduct curve fitting against five reference distributions (exponential, Weibull, Pareto, log-normal and gamma) and we conduct Kolmogorov-Smirnov (KS) tests to see which is the best fit.

We report from our modeling results the p-values and D-values (in Tables 3.6), and the distribution parameters (in Table 3.7). **The Weibull and Gamma distributions provide best-fits for both peak value and maturity at the peak. The growth rate can be best approximated with the Log-normal distribution and for the decay rate Weibull, Log-normal, and Gamma are close approximations**.

Table 3.6: p-values and D-values resulting from KS tests conducted for the parameters of the evolution model over five reference distributions: exponential, Weibull, Pareto, lognormal and gamma.

Series	Exponential		Weibull		Pareto		LogNormal		Gamma	
	p-val	d-val	p-val	d-val	p-val	d-val	p-val	d-val	p-val	d-val
peak value	0.334	0.1	0.426	0.05	0	0.58	0.299	0.11	0.406	0.06
growth rate	0.063	0.24	0.354	0.1	0	0.55	0.359	0.07	0.282	0.15
decay rate	0.116	0.22	0.399	0.07	0	0.54	0.355	0.08	0.372	0.09
maturity at peak	0.261	0.12	0.445	0.07	0	0.71	0.399	0.07	0.445	0.05

The Weibull and gamma distributions provide the best fit for both peak value and maturity at the peak. The growth rate can be best approximated with the lognormal distribution and for the decay rate Weibull, lognormal and gamma are close approximations.

The linear model provides reasonably good D-values. However, modeling the other classes may not be as accurate. Therefore, we consider extending our model using time series analysis [58]. According to the time series analysis theory, we study the evolution patterns under three components (the trend, seasonal / cyclical and irregular components). The irregular component covers the bursts that some of the applications display during their evolution. Naturally, the bursts can either be real or come from reading errors. The chain through which we collected data (described at the beginning of this subsection), allows a number of reporting errors. We consider that system wide events (such as stagnation periods or sudden drops) can be an indication of a reporting error. However, bursts that affect only applications from one developer can derive from promotions or campaigns performed by the developer leading us to think of possible correlations. An algorithm for removing seasonality was shown under characterization. Finally, removing the irregular and seasonal components leave the trend component, which can be approximated with better results by the linear model.

Table 3.7: Parameters for the reference distributions that provide best fit for each of the parameters of the evolution model (see subsection 3.3.2).

Series	Exp(μ_e)	Wbl(k_w, λ)	LogN(μ_l, σ)	Gam(k_g, θ)
p	$4.36 \cdot 10^6$	$0.87\ 4.04 \cdot 10^6$	14.58 1.33	$0.83\ 5.26 \cdot 10^6$
m	50,792	0.66 35,256	9.68 1.62	0.54 93,242
n	8,409	0.68 6,382	7.96 1.66	0.58 14,619
r	19	1.37 21	2.64 0.85	1.76 10

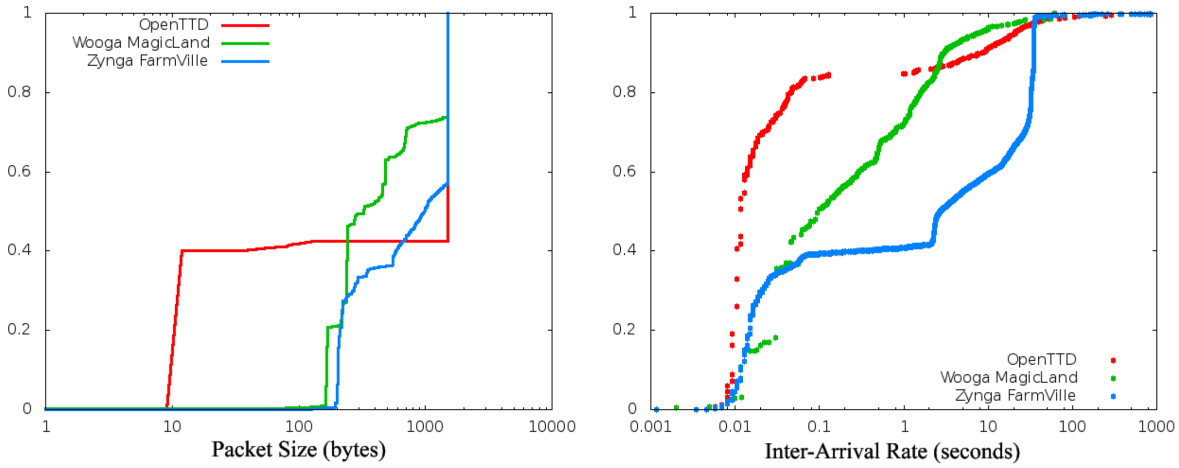


Figure 3.9: Packet sizes (left) and interarrival-times (right) for some of the applications in our datasets

3.4.3 Packet-Level Traces

In this section, we characterize and model the traces from *facebook-1* and *openttd* datasets using the packet sizes and inter-arrival rates. The data regarding Packet-Level Traces for web-based applications was processed together with Emanuel Dias, from Delft University of Technology.

Figure 3.9 depicts the packet sizes (left) and the inter-arrival rate (right) we obtain for some of the applications in our datasets. It is important to note that Farmville and MagicLand are web-based applications, from different developers and that OpenTTD is a tightly-coupled simulation, and although manufactured by different developers, with different technologies, there are some similarities.

Both packet sizes and inter-arrival times are represented on the log scale, which shows that both differ in value with orders of magnitude. It can be noted how most of the packets for all applications cluster around two fixed sizes. There are many small packets (10 bytes for OpenTTD, 200 bytes for web-based applications), which are probably control packets, and many large packets (the MTU size of 1500 bytes denotes messages that need to be split into multiple packets at Ethernet level), which are probably content packets.

Possibly a consequence of status updates, the packet inter-arrival time for web-based applications has an prominent mode around 2s (corresponding to packet size 200 bytes). For OpenTTD, similar modes appear also for similar reasons in FPS games, with frequency of about 40ms and size of 25–80 bytes. They also appear in MMORPG games (50–250ms and 15–30 bytes), and in some RTS games (200–250ms and 8–40 bytes).

Table 3.8: Parameters for the reference distributions (E–exponential, W–Weibull, Ln–log-normal, and G–gamma) that provide best fits for the various empirical distributions. Large values are expressed in thousands and suffixed by K, or in millions and suffixed by M.

Model element	Best fit (parameter values)	Min	1st Q	Median	Mean	3rd Q	Max
Packet inter-arrival time [s]	Ln($\mu_l = 0.28$, $\sigma = 1.95$)	0.01	0.41	1.84	-	4.01	-
Packet size, split [B]	W($k_w = 520.71$, $\lambda = 1.46$)	0	204	242	466	685	MTU
Packet size, un-split [B]	Ln($\mu_l = 7.37$, $\sigma = 1.95$)	0	223	988	12.5K	10.3K	1.04M
Game object size, Flash only [B]	W($k_w = 80,459.72$, $\lambda = 0.81$)	1,004	39K	44.9K	93.5K	81.9K	1.04M
Game object traffic, Flash:All [%]	-	4.3	-	78	64	-	95

Media sizes in web-based online social applications are still very small, from 1KB to about 1MB, with an average of less than 100KB. Flash objects account on average for about two-thirds of game content.

To model the packet size and inter-arrival times, we conduct curve fitting through the Maximum Likelihood Estimation method. We compare the two empirical distributions with four standard distributions and summarize in Table 3.8 our findings. We also segment the packet size metric depending on the object type, and we emphasize the percentage of the Flash files for web-based applications, which represents most of the graphical content that evolves as developers improve the look and feel of their applications.

3.4.4 State Transition Diagram

We analyze the data we collected for the OpenTTD application, using a Sony Vaio 4-core @2.3GHz running Ubuntu 10.04 (*laptop*) as server, and a Asus TF101 2-core @1GHz running Android 4.0.3 (*tablet*) as client. For each of the five stages in the processing loop we report statistical results in Table 3.9.

Table 3.9: Statistics on average running time ($\overline{T}$), maximum running time (T_{max}) and quantity of output data (Q) for the five stages in the loop.

	tablet			laptop		
	$\overline{T}$ [ms]	T_{max} [ms]	Q [bytes]	$\overline{T}$ [ms]	T_{max} [ms]	Q [bytes]
human input	1.2	58	40	0.7	48	40
AI input	48.2	973	40	4.3	209	40
synchronization	188.5	2446	120	101.9	4576	120
simulation	9.3	211	1068576	2.6	283	1068576
rendering	5.7	726	4096000	13.7	135	5206720

As expected, the time values on the tablet are larger than the time values on the laptop for almost all the stages. Rendering takes less time on average, probably because of the lower resolution and the dedicated GPU on the tablet. It is interesting to note that processing time for AI input collection differs on average with an order of magnitude. Simulation on the other hand, although more consuming on the tablet, exhibits a lesser difference, probably because it is optimized by splitting the processing over several iterations.

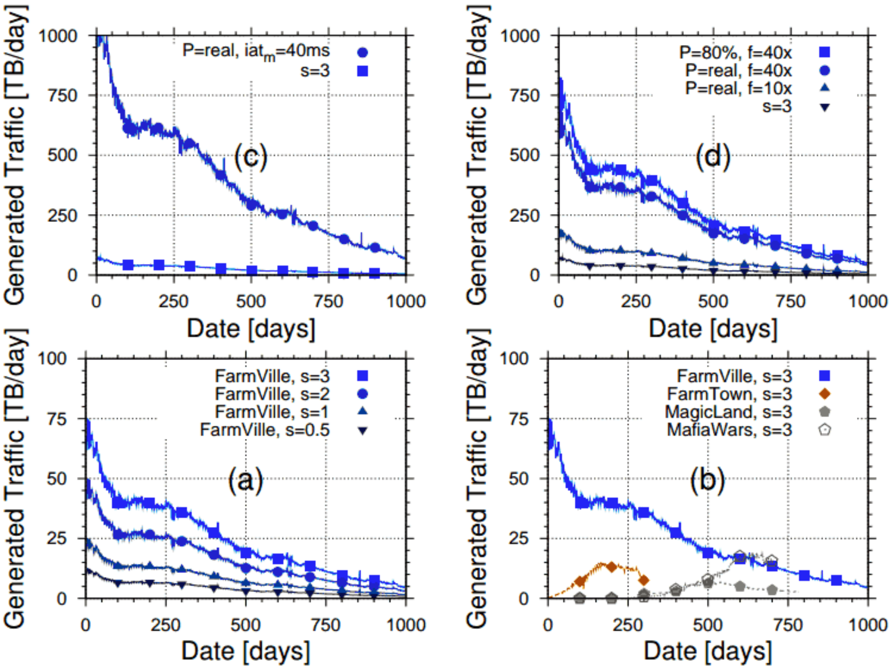


Figure 3.10: Examples of traffic-related applications: (a) traffic vs number of sessions $s \in [0.5, 3]$; (b) traffic for various applications; (c) traffic after a hypothetical increase of the packet arrival frequency; (d) traffic after a hypothetical increase in the size of Flash files ($f \in [10\times, 40\times]$) and in the fraction of media files among online social application objects ($P \in [66\%, 80\%]$).

3.5 On the use of our model

The workload evolution model for applications has various uses. For a company that develops an OSG similar to others, our model as a whole provides the tools to predict network capacity requirements, based on assumptions about the online social application's relative ranking versus the competition. Our model is also useful for companies that develop novel applications or when data related to the competition is not available: then, the needed network capacity can be based on targets set by the developer about the online social application's initial growth and peaks. As a third general example, when coupled with a low-level packet generator, our model provides the tools for tuning existing and designing new OSG architectures, by generating the input needed for simulations.

We present, in the following, several specific applications.

3.5.1 Traffic-Related applications

What is the traffic generated by applications? To give a reasonable estimate, practitioners can explore various parameters of our model. s, the number of sessions played by each player, on average, per day, is also needed to obtain actual estimates; we use in this study a realistic range for s: $[0.5, 3]$. Figures 3.10a,b depict the traffic of various applications. Even for the low value $s = 0.5$, the total daily traffic we estimate for FarmVille, after normalizing by number of players, is much higher than for OSN gifting applications. We explain this by the different application requirements, e.g., FarmVille, as a online social application, has much higher packet inter-arrival rates.

How would applications traffic change if applications increase pace or improve their multimedia quality? Figures 3.10c,d shows estimates for a transition to FPS-like packet inter-arrival times (median: 40ms) and for a transition to much larger content sizes (Flash files scale up 10–40 times and/or become more present). For an application of FarmVille's popularity, the traffic could increase significantly, up to 1PB/day.

3.5.2 Evolution Typology-Related applications

The evolution typologies described in Section 3.4.2 are useful in running networked online social application operations. Stationary (Class-1) applications are relatively easy to maintain, requiring only basic spare-capacity planning and management. Single-peak (Class-2) applications are favorable for predictive provisioning of resources for the infrastructure on which they operate. However, the periods of opposite effect may lead to inefficient resource provisioning which, especially when these periods occur close to the peak, may lead to significant unnecessary costs. Class-3-to-5 applications represent difficult cases for resource provisioning, requiring not only adequate predictions of when the burst(s) will occur, but also finding a provider that can offer the large amount of needed resources without a clear understanding of the peak. Even when a cloud provider can offer the necessary networked resources, these OSG classes require constant attention to avoid missing peaks and thus causing unnecessary costs, and may require a special commercial agreement between the OSG operator and the resource provider.

Predicting whether an application will be successful (predicting the peak and lifetime) is a major focus of any online social application operator. Future predictive algorithms can use the CDF of maturity at the peak, depicted in Figure 3.8d, indicates that most applications have a peak within 15% of their lifetime and 85% have a peak within their first third of their lifetime. The models for each evolution pattern could be useful for predicting application lifetime and for designing future prediction algorithms.

Chapter 4

Offloading Mechanisms Analysis

People use mobile devices daily in activities ranging from entertainment to solving professional tasks. Mobile applications span *a vast application-domain*, being developed for various purposes, such as gaming, multimedia streaming, travel, communication, etc. Many of these types of applications rely on connectivity and on data stored remotely. Also, many of them make a lot of use of the high computation power of mobile devices. Among this generous application space, we identify several types of applications that would benefit from offloading:

- applications that are computational intensive, like Chess and other low-response-time games: these apps are fit for remote processing because users do not expect quick responses from the program

- applications that rely on data from server side, such as Shazam, the program that recognizes songs by comparing samples with a vast cloud database: the amount of data these apps use is prohibitive to a single device functionality

- applications with a particular, pipeline-based processing, such as image processing applications: these applications are suitable for offloading in various degrees, only parts of the pipeline can be offloaded, depending on the conditions at hand

- applications that already interact with the cloud, such as m-commerce applications: these apps are already making use of the communication capabilities, so employing other forms of offloading are just a matter of fine-tuning

In this chapter, we investigate Communication Adaptation and Offloading for distributed applications spanning the mobile device and custom hardware extensions, such as sensor devices and home automation networks. We also investigate Computation Adaptation and Offloading for loop-based applications, with a focus on video processing (using an augmented reality application) and video rendering applications (using a popular simulation game). Finally, we propose an operational analysis to represent mathematically all the offloading mechanisms identified in Section 2.4.2, for any loop-based application.

4.1 Communication Adaptation and Offloading

As increasingly more intelligent interconnected devices surround us each day, to fulfill the vision of Ubiquitous Computing [1], several challenges and research opportunities arise. Smaller devices, like sensor nodes in Wireless Sensor Networks, are usually limited in terms of computation power and battery supply, so they rely on communication, through WiFi, Bluetooth and other emerging technologies. In mobile devices more than a third of the power consumption is spent on communication tasks, while in small sensors more than a half is spent on communication [59]. Devices can be made smaller and smaller, but battery cells usually cannot.

In this section we explore communication adaptation and offloading mechanisms with a focus on interconnecting mobile devices with even more limited sensor devices. First, we investigate communication adaptation in a system built by our team, consisting of a sensor device that monitors air quality and a mobile device that performs periodic queries to read sensor values. Second, we experiment with communication offloading to a dongle, a small device that connects to the mobile through USB, and enables communication with a home automation network through the ZigBee communication protocol.

4.1.1 Adaptive Query Algorithm for Location-Based Applications

We develop a system, called PollutionTrack, consisting of a smartphone and an embedded device designed to measure air quality [60]. Figure 4.1 shows the sensor device and a map on the mobile device with an overlay of information collected from the sensor device. Communication between the mobile phone and the sensor device needs to be effective and power efficient in order to have a minimum effect on the phone's recharge cycles. Our approach to this challenge is to reduce the power consumption of the device using an adaptive query algorithm which minimizes the energy used in data transfers. The algorithm gives great advantages to location-aware projects by reducing data communication when it is not needed.

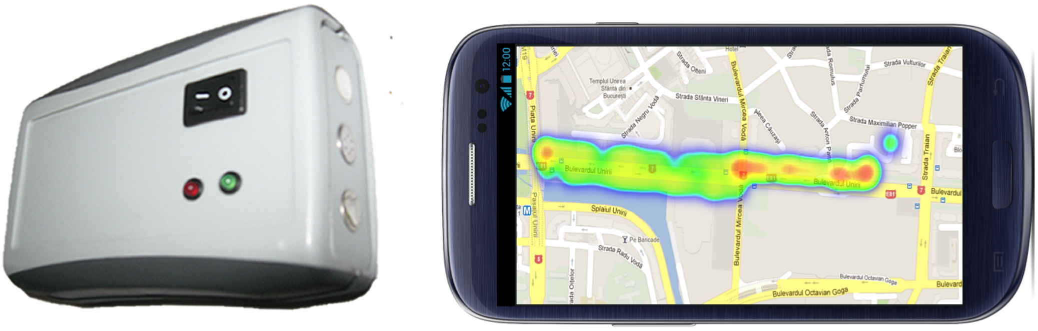


Figure 4.1: Sensor device (left) and a map overlay (right) with information from the sensor device.

System Design and Implementation

The PollutionTrack system consists of an Android application interconnected with an embedded sensor device, that measures air quality with a NO sensor, a CO sensor and a particle count sensor. The sensor device was designed and built by Dan Tudose and the Android application was implemented in collaboration with Daniel Rizea [61], both from University Politehnica of Bucharest. The application on the smartphone needs to query the sensor device using Bluetooth in order to collect pollution data, as shown in Figure 4.2. On the mobile device, a sensor service periodically sends a short command to the sensor device, depicted in the figure as the letter "‘a"’. The sensor responds with a package consisting of the letter "‘ "’, for consistency checks, and four numbers, representing the values from the three sensors and the battery level of the sensor device.

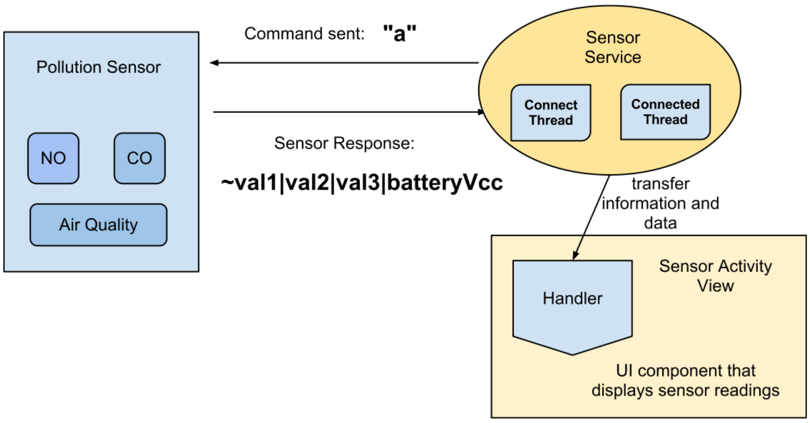


Figure 4.2: General architecture and communication mechanism

The communication between the mobile phone and the sensor device needs to be effective and power efficient in order to have a minimum effect on the phone's recharge cycles. Our approach to minimize power consumption is to switch on the Bluetooth adapter only when a query must be done. We will show the effectiveness of this method in the next section.

PollutionTrack is designed to work in either passive mode or active mode. In *passive mode* the user is not informed when the pollution levels are increasing. This mode can be utilized when the user does not want to be actively informed of pollution danger levels and only wants to collect air pollution data. In this case, the update period, cool down time and wakeup parameters can be lowered. The location will have the biggest impact on query intervals and pollution data values will have a small contribution in setting the update periods. The *active mode* of the system is utilized when the user wants to be actively informed whenever specific pollution levels are exceeded. In this case the adaptive algorithm is more aggressive and will have a smaller wakeup time interval. The system will make more queries whenever an increasing pattern of pollution values is detected. This being the case, the query period of the sensor will be heavily influenced by the pollution information and not so much by location changes.

The Adaptive Query Algorithm

We propose an Adaptive Algorithm that changes the update time interval based on the previous user location and the previous update time. This approach is suitable for location-aware applications by reducing data communication when it is not needed.

For example, let us consider that a user remains in a location for about 1 hour. In a normal approach 60 queries will be made for the device (assuming a 1 minute period between queries). In our adaptive approach the second time the application wakes up it will compare the last location with the previous one and will increase the query interval by doubling the previous time interval. Only 6 sensor queries will be executed. Let us assume now that the service battery usage for 1 hour and 60 requests is about 4% of the battery capacity, then our algorithm will have a total of 0.4% battery usage over 1 hour. The improvement is significant and is needed in order to make the application popular among users.

It is important to note that, when applying the algorithm to other applications, parameters such as the reference query time interval and the cool down time can vary according to the application's purpose and needs. The variations of these parameters influence the total power consumption of the Bluetooth module. In the remainder of this section, we give a formal presentation of the adaptive algorithm tuned for the PollutionTrack application.

The update time interval is dynamically calculated and uses the following formula:

$$T_u(mode) = T_b(c_{ld}(mode) + c_{dd}(mode)) \tag{4.1}$$

, where:

- T$_u$: adjusted query time interval
- T$_b$: reference query time interval
- c$_{ld}$: location dependent coefficient
- c$_{dd}$: data dependent coefficient

The location dependent coefficient and the data dependent coefficient are computed separately and their values follow the observations made regarding the system characteristics in passive and active mode.

The location dependent coefficient is computed based on the current location loc^{i+1} and previous location loc^i. When in passive mode Equation 4.2 is used, and when in active mode Equation 4.3 is used, where α and β are adjustable parameters. Both equations work on the same principle: if the location stays the same, then the time between queries is increased, which means the smartphone will issue fewer queries and less power will be consumed. If the location changes then the time between queries is reduced, which will enable the sensor to collect fresh data. This behavior is based on the fact that pollution information does not change radically in a small period of time and in the same location. The diference between the two modes is however in the ratio with which the coefficient is updated. The active mode of the algorithm is used for dangerous environments where pollution levels can grow rapidly in the same location and over a few minutes.

$$c_{ld}^{i+1}(passive) = \begin{cases} \alpha c_{ld}^{i}(mode^{i}) & \text{if } loc^{i+1} = loc^{i} \\ \frac{1}{\alpha} c_{ld}^{i}(mode^{i}) & \text{if } loc^{i+1}! = loc^{i} \end{cases} \tag{4.2}$$

$$c_{ld}^{i+1}(active) = \begin{cases} 3\beta c_{ld}^{i}(mode^{i}) & \text{if } loc^{i+1} = loc^{i} \\ \frac{2}{3}\beta c_{ld}^{i}(mode^{i}) & \text{if } loc^{i+1}! = loc^{i} \end{cases} \tag{4.3}$$

To compute the data dependent coefficient, we hold a queue with the last n recorded values. Based on these and the current value from the sensor, we determine how fast the pollution data changes. We compute the change rate with $r = \sum \frac{v-D_i}{v}$, where n is the window size, v is the current value and D is an array of size n with recently collected data. The size of the window should vary according to different needs. In our experiments we use a size of 5 values, as this should cover timeframes of more than one minute, given the sampling period which, in our case is 15 seconds. If the change rate is positive, there is a growth in the values, so the queries should be made more often, at a rate directly proportional with the growth rate. If it is negative, the queries should occur less often. The data dependent coefficient also depends on the mode: in active mode (Equation 4.5) the effect of the change rate is twice the effect in passive mode (Equation 4.4).

$$c_{dd}(passive) = \frac{\lambda}{\sum \frac{v-D_i}{v}} \tag{4.4}$$

$$c_{dd}(active) = 2\frac{\lambda}{\sum \frac{v-D_i}{v}} \tag{4.5}$$

Testing and Performance Evaluation

We conduct functional testing and performance evaluation of our solution, using our custom sensor device and a HTC Nexus One, running Android 2.2. Some relevant hardware characteristics are its Qualcomm Snapdragon CPU at 1GHz and the Bluetooth module with Bluetooth 2.0 + EDR. For testing, we have used both the sensor device and a Java computer application that we developed, which behaves exactly as the pollution sensor, having the same Bluetooth communication protocol. With this program we can emulate the behavior of the sensor and vary the sensors data to test the data dependent function.

Each test involves running the application for 300 seconds. Most of the performance tests were run while remaining in the same location, as this is the maximum efficiency case for the algorithm. We study three query strategies: Permanent Connection (*Permanent*), Fixed Time Query Interval (*Fixed*) and Adaptive Query (*Adaptive*). In the former, the smartphone establishes a permanent connection with the sensor device. This is demanding in terms of energy consumption from the CPU and from the Bluetooth module. For Fixed Time Interval Query the CPU and Bluetooth module are put into connection state, where power consumption is the largest only when a data query is needed. The Adaptive Query uses our Adaptive Algorithm to dynamically select a query time interval.

Power consumption on the CPU has been recorded using PowerTutor [25], an Android application that estimates power consumption on different hardware components based on an energy model that the authors derived in their studies. We also estimate the power consumption on the communication module using this simple model:

$$E_{total} = P_{conn} * T_{conn} + P_{trans} * T_{trans} + P_{on} * T_{on} \tag{4.6}$$

Based on Formula 4.6 we can calculate the estimated power consumption for the three query strategies. For all the three cases we have a T_{trans} time that is the same for Permanent Connection and Fixed Time Query Interval but is smaller for Adaptive Algorithm. In the case of Permanent Connection we have no T_{on} time because the Bluetooth is always connected. For Fixed Query Interval we have a T_{conn} time when the Bluetooth is in connected state. After the data is transmitted the Bluetooth connection is terminated, thus saving energy. In the Adaptive Algorithm the Bluetooth connection time varies, adapting the query time based on location and environment data. The measurements done in [62] show that a Bluetooth module in the on state with no connection $P_{on} = 15mW$, in the connected idle state takes $P_{conn} = 65mW$ and in transmission $P_{trans} = 432mW$. In both the Fixed Time Query Interval and Permanent Connection we intend to read values from the sensor at every 15 seconds.

Data transfers are performed on a query basis: the smartphone asks for data from the sensor device, which in turn responds with a single message. Thus, for a single exchange, we have 2 bytes sent from the smartphone and 21 bytes received. Figure 4.3 shows the variation power consumption on the CPU for the three query strategies during the 300 seconds tests. Table 4.1 shows the total estimated power consumption on the Bluetooth module and on the CPU.

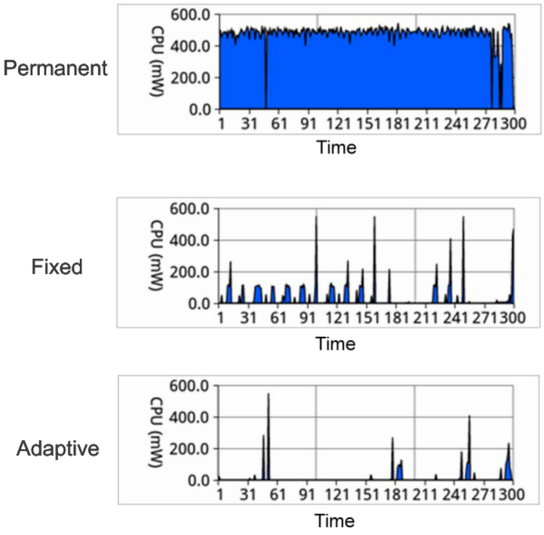


Figure 4.3: CPU energy consumption

Table 4.1: Estimated total energy consumption on CPU and on WiFi for the three query strategies, during 300 seconds tests.

	CPU Energy (mJ)	Data transfer (bytes)	Comm. Energy (mJ)
Permanent	136,133	21 x 20 = 420	28,740
Fixed	5,465	21 x 20 = 420	14,180
Adaptive	1,310	21 x 4 = 84	6,228

From the results we can see that the Adaptive Algorithm approach is almost 5 times better than the Fixed Time Query Interval and 100 times better than the permanent connection approach. The significant power saving is justified because in 300 seconds, the duration of the tests, we had 20 fixed queries at every 15 seconds in Fixed Time Query Interval mode and only 4 queries in the Adaptive mode (the location was not modified) which makes the Adaptive Algorithm mode to have 5 times less queries than Fixed Time interval.

T_{trans} is less for the Adaptive Algorithm than in the other two cases because the adaptive query algorithm varies the query interval. In each of our 300 second tests we have 20 requests at intervals of 15 seconds. The transmission time takes approximately 1 second per query. $T_{conn} = 300$ seconds and T_{trans} is 15 seconds. $T_{on} = 285$. With the help of the adaptive algorithm, the number of queries goes down to 4 queries in 300 seconds.

Conclusion

We have studied power consumption in a system consisting of a smartphone and an embedded device designed to measure air quality. The system has two functioning modes: passive and aggressive. These two modes influence how the adaptive algorithm performs. In passive mode, the location factor has a greater impact on the query interval. In aggressive mode the recorded data varies the parameters used in the algorithm.

We have done measurements and estimations to compare power consumption dedicated to CPU and Bluetooth usage in three cases: using a permanent connection, using a fixed time query interval and using our adaptive algorithm. In terms of CPU power consumption, we find that our algorithm is 5 times more efficient than the Fixed Time Query Interval strategy, that is proportional with the number of queries. In terms of Bluetooth power consumption, reducing the transmission time and the connected idle time, the Adaptive Algorithm is approximately twice more efficient than the Fixed Time Query Interval strategy, which in turn is approximately twice more efficient than the Permanent Connection.

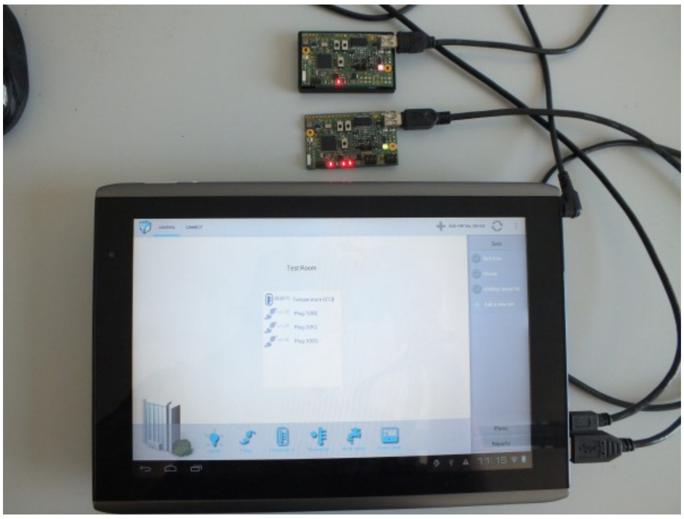
The adaptive algorithm is suited for location-aware applications, leveraging the fact that new queries are not necessary if the location has not changed significantly. We find that considerate querying can lead to energy consumption 20 times smaller than in the case of a permanent connection. We plan to improve the algorithm in our further studies, by determining velocity and using machine learning techniques in order to alter the query period.

4.1.2 Offloading Communication to Custom Hardware Extensions

Mobile devices can communicate with a home automation network through an Internet gateway, but cannot directly communicate with devices in the network, which usually implement low power communication protocols like IEEE 802.15.4. Some research was conducted in using smartphones along with IEEE 802.15.4 [63][64]. However, to the best of our knowledge, there is no work that investigates this usage in the context of home automation. We investigate the use of multiple communication channels, such as the TCP channel, that uses WiFi to connect to a gateway, and the USB channel, that can connect to a device on the home automation network through an USB dongle.

Our main contribution is three-fold:

Figure 4.4: Test with a client device. The test dongle (right on top of the tablet) uses the same platform as the network device (top board).



- We investigate different ways to connect, to an Android device, a dongle capable of mediating two-way IEEE 802.15.4 communication (Figure 4.4)
- We propose a general client architecture and an Android implementation, with several abstraction layers, that can support multiple communication channels and enable other developers to offer different user interfaces to our communication system, while hiding the home automation system complexity
- We estimate energy consumption for WiFi transfers for typical home automation scenarios

We find that using an USB host is the most promising solution, among the ones we investigated, in connecting to the mobile device a dongle capable of two-way IEEE 802.15.4 communication. We propose three abstraction layers to unify local and remote access, different communication channels, as well as various ways of offering the functionality to multiple user interfaces. Our energy estimation for three use cases, typical for home automation, indicates that it is worth using low-power protocols such as the IEEE 802.15.4, especially for issuing commands with small payloads, like turning on and off the lights.

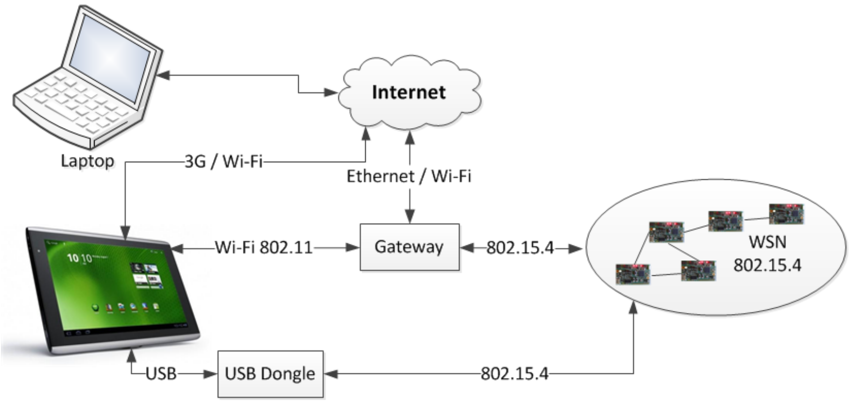
System Design and Implementation

The home automation system used in our work (Figure 4.5) consists of:

- Various home automation devices interconnected in a wireless sensor network, each running the ZigBee stack; for our tests, we have used a couple of ZigBee nodes with LED lights simulating a conventional home light;
- A gateway at the edge of the network, in the form of a PC, which has several functions, such as providing a two-way interface with the Internet and caching sensor data in a local database;
- One or more client devices that provide the user with status updates and command functions, while ensuring proper security and privacy considerations, as well as

concurrent access; the clients should be able to access the home automation network either through the Internet gateway or the USB dongle.

Figure 4.5: Home automation system architecture



Using the network gateway to issue a command consists of: (1) creating a data package on the client, denoting the type of command, the extended address of the destination device, and the actual payload, as described by the ZigBee specification (simplest commands are formed with 48 bytes); (2) sending this data package to the network gateway, with the overhead given by network encapsulation; (3) forwarding the command through the ZigBee network to the destination device. In contrast, a client device with ZigBee capabilities, can simply connect to the destination device, if in range and send the command directly.

ZigBee is the communication protocol of choice in home automation networks because, as shown in Table 4.2, it is an energy efficient protocol, ideal for small messages and routing information through the network. Considering that usually Bluetooth modules operate at about 1.8V and ZigBee modules at 3.3V, Bluetooth power consumption is within the same range as ZigBee power consumption. In contrast, Wi-Fi and UWB take up to 230mA for transmitting and receiving data. While Wi-Fi and UWB have higher power consumption levels, they also have a considerably larger bandwidth, which offers them a very low ratio of energy/transferred MB. Bluetooth has a medium bandwidth, and ZigBee has the lowest bandwidth, making it ideal for small messages.

Modern mobile devices have embedded modules for several wireless communication technologies, such as WiFi, UMTS and Bluetooth. However these are not applicable for home automation devices. To enable the mobile device with a ZigBee module, an extension in the form of the dongle is needed. To identify an appropriate communication method between the mobile device and the dongle, we compare several candidate technologies (detailed in Table 4.3):

- *Audio jack*: the audio jack is a universal connectivity port featured on all modern mobile devices. Signal is transmitted in an analog form, as it is primarily designed for audio signals, over up to four channels. Digital data can be transmitted by encoding the signal [65] or by using Frequency Shift Keying [66]. However, the bitrate is significantly low and its limited ability to gather power from the mobile device makes it prohibitive for use as a dongle;
- *USB*: Universal Serial Bus is an asymmetric communication protocol, which assumes

Table 4.2: A comparison of communication technologies

	ZigBee	WiFi	Bluetooth
Bandwidth Range	IEEE 802.15.4 250 Kbps 10-100 meters	IEEE 802.11a/b/g 54 Mbps 50-100 meters	IEEE 802.15.1 1 Mbps 10 meters
Topology	ad-hoc, star or mesh	point to access point	ad-hoc, small networks
Frequency	868 MHz (Europe) 900-928 MHz (NA) 2.4 GHz (world)	2.4 and 5 GHz	2.4 GHz
Coexistence	dynamic freq. selection	dynamic freq. selection	adaptive freq. hopping
Power	25mA TX, 27mA RX, standby 3uA	219mA TX, 217 mA RX, standby 20mA	57mA TX, 47mA RX, standby 0.2mA
Typical apps	industrial control sensor networks	Internet access	headsets, file transfer

the existence in the communication process of a host and a client. Thus, USB communication for Android comes in two flavors: USB host, when the mobile device plays the role of the host, and USB accessory mode, when the dongle plays the role of the host and is called an accessory;

- *Bluetooth*: is a communication protocol with a long tradition, usually intended for file transfers on a short range; now at its fourth iteration, the Bluetooth protocol features profiles for low power consumption as it is usually used on devices with energy constraints, such as those involved in home automation.

Table 4.3: Methods to connect a dongle to a smartphone

	Audio Jack	**USB Host**	**Accessory mode**	**Bluetooth**
Two-way serial	Yes (analog)	Yes	Yes	Yes
Power Devices	Yes (7.4 mW) **Most phones and tablets (requires TRRS jack)**	Yes **Tablets (Android 3.1 otherwise recompile)**	No **Phones & tablets (Android 2.3.4 and 3.1)**	Not needed **All phones and tablets (Bluetooth 2.1)**
Pros	All devices No changes to kernel	Power Reliable serial	**Works on phones, not only tablets**	All devices No kernel change
Cons	Analog unreliable little power	only some devices different connectors	no power dedicated chip different connectors	not really a dongle

While the audio jack and Bluetooth are the only truly universal connectivity ports, they lack the performance and reliability of the USB connection. On the other hand not all Android devices have a USB port and, those that have one, can have one of the several variants of USB ports, which makes it a challenge for the dongle designers. Using the Android device in accessory mode is an interesting approach, promoted by Google through the Accessory Development Kit, dedicated especially to the older phones, which do not have the USB Host mode. However, having the mobile device act as a client would mean that the dongle would have to power the bus, which is completely unfeasible in our case,

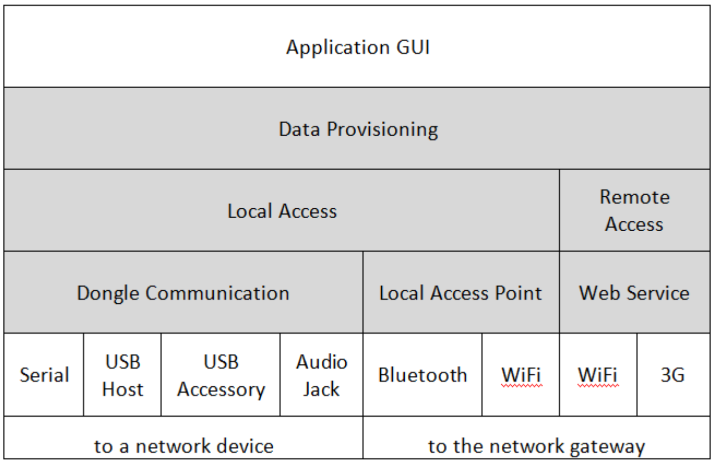


Figure 4.6: General client architecture

due to the form factor restrictions. USB is our method of choice because it conforms with all our three criteria:

- *Two-way serial*: the ability to communicate in both directions, in a serial manner
- *Supported mobile devices*: supported by many off-the-shelf devices
- *Power consumption*: a dongle without its own power supply is preferable, due to size factor and convenience; thus, it is necessary to draw power from the mobile device.

As part of our collaboration with the University of Applied Sciences in Dresden, we have acquired a communication dongle developed by ZigPos GmbH, which connects to the mobile device via USB following our specifications. This dongle features an FTDI chip for USB communication, connected to the phone, and an IEEE 802.15.4 module for ZigBee communication, with the home automation devices.

For the client design we propose the general client architecture in Figure 4.6, with the purpose of supporting all the communication methods described in our analysis so far.

For the Android client implementation we only use two communication methods: USB Host to connect the dongle, and wireless for remote access. The application has a hierarchical structure (Figure 4.7), which interposes between the GUI and the communication modules three layers of abstraction, so that it can adapt to a variety of devices:

- *Data Provisioning Layer*: provides an abstraction over all types of communication methods, either local, via a dongle, or remote, via a web server; the API exposed by this layer has the essential function of enabling various developers to propose different GUIs over a single mobile application structure;
- *Access Layer*: unifies all the local access methods to communicate with the home automation network from within the premises, and, separately, all the remote methods that are used to communicate with the home automation network remotely via internet; local access is usually trustworthy while remote access requires some additional form of security and stronger authentication; the API exposed by this layer is used by the Data Provisioning Layer to provide a single API to the GUI;

- *Communication Channel Layer*: uses any of the communication modules described in Section II, either wired, such as serial, USB, audio jack, or wireless, such as Bluetooth, WiFi; all the wired communication channels, as well as the Bluetooth, are used to communicate with a IEEE 802.15.4 capable device, such as the USB dongle we used in our tests, that in turn connects dirrectly with a device from the home automation network; the web service query channel, as well as the local WiFi module can be used to communicate with the home automation network via a server, through internet and local area network respectively.

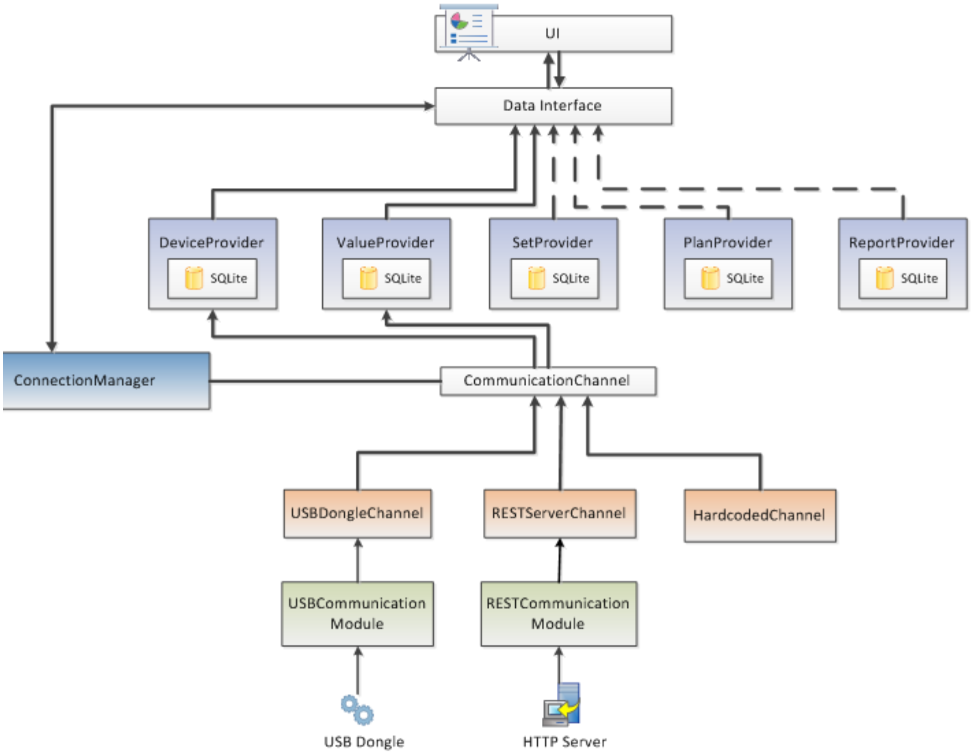


Figure 4.7: Android application structure based on the abstraction layers proposed in the general client architecture.

We implement an Android application that uses general object-oriented programming concepts and particular Android features to implement communication with multiple channels, following the three proposed abstraction layers.

The Data Provisioning Layer uses Content Providers, which are Android specific constructs meant to encapsulate data and serve it to multiple processes. In our implementation, each of the five Content Providers uses a local SQLite database to store data regarding one key concept of a generic home automation network. The *device* can be something like a light source or a thermostat, and has several characteristics, such as location and network address. Each device can have either numeric or Boolean (on/off) *values*. A *set* of devices is a manual grouping of devices that need to be operated together for a complex functionality, such as putting the whole house in stand-by when going on holiday. The *plan* associates certain values to certain devices at a given timestamp. The *report* is a collection of aggregated values, either sampled or averaged. The Data Provisioning Layer also provides caching of data on the local storage of the device. Values are kept on the device for a week, after which they are only available on the server.

In this stage, the Access Layer is poorly represented. We currently do not make any difference between local access and remote access. Although security can be provided with traditional means, such as using HTTPS for communication over WiFi, the role of the Access Layer is crucial in a real-world implementation.

The Communication Channel interface is an abstract class that is meant to serve the Communication Channel Layer in providing communication capability with various modules. It describes the abstract methods of reading and writing data. In this stage, it has three implementations: (1) the USB Dongle Channel uses the USB Communication Module to conduct serial communication with the dongle in order to read and write data, for local access; (2) the REST Server Channel uses the REST Communication Module to conduct wireless communication with a network gateway, for remote access conforming with REST principles; (3) the Hardcoded Channel simply provides mock-up data without actually communicating with the exterior and was used in testing.

The Connection Manager uses a Service, which is an Android specific construct, meant to perform long-running operations in the background and provide inter-process communication. In our case, it monitors the availability of communication channels and can be used by the upper layers in the architecture for operations such as selecting an active communication channel. This option is necessary because, if more communication channels are available in the same time, only one can actually be used. If both the USB Dongle Channel and the REST Server Channel have connectivity, the USB Dongle Channel is preferred by default.

On top of this structure, we propose a UI focused on *usability*. The UI is based on several concepts that describe basic and complex entities of a home automation system. Basic entities are *devices* and their *values*, as described earlier in this section. Among the complex entities, the *profile* groups a set of devices and their desired values, and the profile's state, which is either active or inactive. We also use some time related concepts: the *event*, which has the role of associating a timestamp with a device and a value, like the lights were turned off, and a *schedule*, which is basically a set of events. These concepts can model most of the use cases envisioned for home automation. For example, when going out on holiday, the user can activate a stand-by profile, which passes all the associated devices to low-power values, such as turning off the heating on all the electric plugs. We use Content Providers and a Service, which provide inter-process communication, to provide a loose coupling between the UI and the rest of the application structure. Thus, the UI can be easily extended by third-parties, who can write applications that use the lower levels of our application.

Testing and Performance Evaluation

We test the application with two client devices running Android 4.0, an Acer Iconia A500 tablet and an Asus Transformer TF101 tablet. We also use a mock home automation network consisting of a PC gateway and two ZigBee devices, developed by ZigPos, simulating conventional home lights (Figure 4.4). We identify and test several use-cases, which may occur during the normal usage of the application:

- *Use Case 1 - toggle command*: in this use case, the user issues a simple command to toggle the lights; according to the ZigBee implementation, this translates into a series of bytes denoting the type of command, the extended address of the destination device, and the actual payload;

- *Use Case 2 - encoded image:* the system allows users to take a photograph of the network device and replace the default icon with the actual image;
- *Use Case 3 - history of values:* the implementation usually keeps a limited cache of values on the device and periodically stores them on larger storage devices; in some rare cases, the users might be interested in detailed historical values registered by the home automation network, in which case the application can retrieve a detailed history in a compact JSON format; we consider this the most demanding, but also the rarest use case.

We estimate the power consumption over these three use cases using WiFi. Although we could not find any datasheet for the WiFi module of neither of the two tablets, we estimate the following data at a voltage of 3V.

Table 4.4: Estimation of energy consumption.

	original size	actual size	link speed	time	current	energy	energy/byte
	[bytes]	[bytes]	[Mbit/s]	[ms]	[mA]	[uJ]	[uJ]
toggle command	48	664	26	0.003	270.40	2.47	0.051
encoded image	57,296	57,912	39	0.177	270.40	143.62	0.003
history of values	2,468,676	2,491,895	52	5.713	270.40	4,634.06	0.002

We use the Android internal API to read values for the actual sizes of the transfers, link speed and current value for the WiFi module in active state. The values reported in Table 4.4 are averages over five readings.

The actual size of the transfer is larger than the original size, as it accounts for headers and failed packet transfers. The link speed varies a lot from one run to another, so the final estimation can actually have quite a big offset. However, we find that the average values of the link speed describe well the expected behavior of a WiFi module, which should work well for large transfers. Thus, we consider the final estimated values in a valid proportion one to another.

Using the data we read, we estimate the time taken by each transfer, and the total energy consumed by each transfer. We also compute the energy/byte, using the total energy and the original size. We use the original size instead of using the actual size, to account for packet headers and failed packet transfers.

Observing the energy/byte in Table 4.4, we conclude that the WiFi module consumes a lot of extra energy for small transfers, such as the ones in Use Case 1, which we expect to be the most numerous in a real deployment. For such transfers, using ZigBee might prove to be energy efficient.

Conclusion

In this paper, we study the integration of mobile devices into home automation systems. We investigate different ways to connect, to an Android device, a dongle capable of mediating two-way IEEE 802.15.4 communication.

We propose a scalable architecture, with three abstraction layers, to unify local and remote access, different communication channels, as well as various ways of offering the functionality to multiple user interfaces. We hide the complexity of the notions involved in the home automation system by including them into a simple, but comprehensive set of related concepts. This simplification is needed to fit as much of the functionality on the limited space offered by a mobile device's display.

We test the application with two client devices running Android 4.0 and a test network consisting of a couple of ZigBee devices and a PC as gateway. We estimate the energy consumption of WiFi transfers over several typical use cases and we conclude that using IEEE 802.15.4 can prove beneficial both in terms of functionality and performance.

4.2 Computation Adaptation and Offloading

In this section we explore computation offloading mechanisms. We first focus on adaptation of quality in a video-processing application and then we employ offloading in a video-rendering application trying to preserve quality.

4.2.1 Performance Adaptation in Video-Processing Applications

A simple camera-enabled smartphone can be used to step into a virtual or augmented world. For example, in a museum, users looking at maps through their mobile phones can see animation depicting troop movements, and when looking at posters with images they can see instead short movies. Augmented reality is based on processing the frames registered by a camera and overlaying rendered objects based on either location information or marker tracking.

We develop an augmented reality application to study the stages through which information needs to pass in order to be displayed to the user (Figure 4.8). We choose QR codes as the starting point of our marker-based tracking system, as they already benefit from a number of implementations for various mobile platforms. The QR codes provide two important features:

- they store compressed information, large enough to implement a client-server system for retrieving information;
- they provide means for easily detecting surfaces, by locating the corners of the code.

We identify the bottlenecks of the system, with the purpose of maximizing its performance and we propose an adaptation mechanism of skipping frames at each stage in order to provide a fast on-screen response. We conclude by measuring the output frame rate, with respect to the hardware specification of the devices.

System Design and Implementation

We design, implement and evaluate an augmented reality application for Android. This study was conducted in collaboration with Alexandru Gherghina [67]. The application's

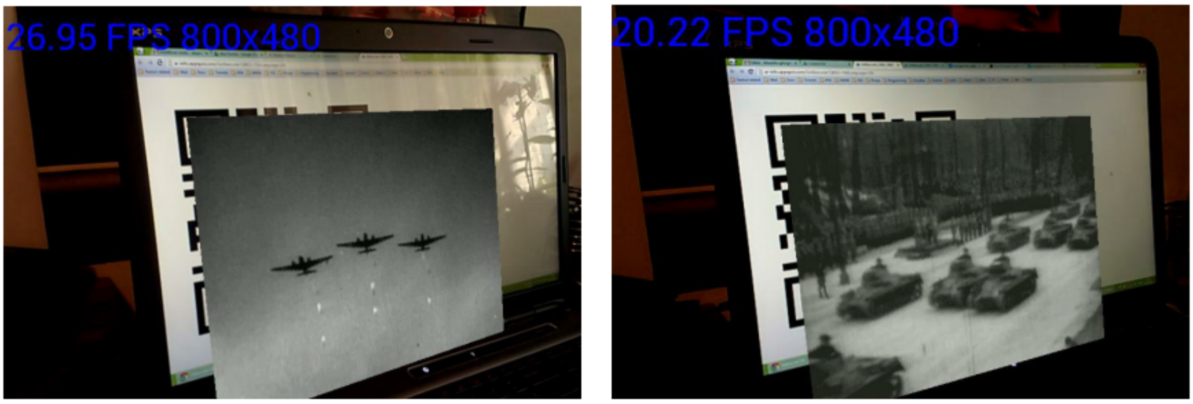


Figure 4.8: Application screenshots, while displaying historical video clips on top of a QR code shown on a screen.

primary requirement is to replace specific markers with images, animations or even movies. For example, it may be used to enrich a person's experience when visiting a museum, or can overlay informational boards with more attractive and content-rich posters.

The implementation uses QR codes as markers for the augmented reality to take place. The QR codes possess the advantage that they contain data that can be used to identify the content, as well as data related to the physical location of the marker. A very dynamic system may be achieved in this way. No marker is programmed directly into the application and it may distinguish between two different markers with ease. Unfortunately QR codes also present a disadvantage: slow recognition and relatively small recognition angles. We consider that the QR code contains the address of the server where the content is stored and the ID of the content that is supposed to be displayed. The necessary data format will be explained when describing the QR code decoding module.

The system is composed of a client and a server. The server's job is to store content, answer to client requests to download content and manage the existing content in the database. The client retrieves content and projects it on the display.

The client application runs in a cycle, depicted in Figure 4.9. This is consistent with the State Transition Diagram of our workload model presented in Section 3.3.4.

The server has a very simple architecture. It contains a series of Java servlets that answer the requests from the clients and enough storage space for all the content. In addition, we are offering a web administration section where content can be added, deleted or modified. The current implementation resides on Google App Engine, a cloud computing platform offered by Google. We choose this platform because it offers an integrated storage system.

The steps in the cycle are represented as if they were sequential. However, to speed up the actual content projection and to make use of the device's processing power, the application relies heavily on separate threads (see Figure 4.10). Therefore, a set of modules has been designed, modules that can run independently from one another.

In addition, to make sure the module that displays the camera preview is not slowed down by other processing, we used a pipeline programming model. Every module reads data from an input buffer and after processing it will write the data into another buffer so that another module can make use of it. For example, when the content downloading module

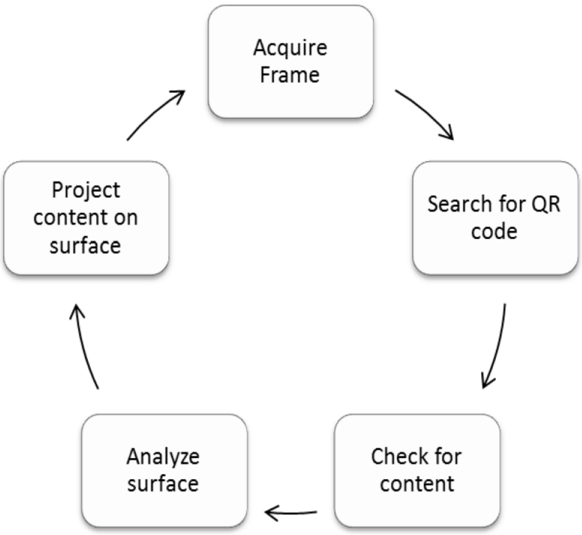


Figure 4.9: Basic client application cycle

has to connect to the server and get the content, the other ones don't have to wait around for it and keep processing whatever data they have.

The modules correspond to the execution steps:

- Image acquisition module: initializes the camera and obtains individual frames;
- QR code reader module: searches for a QR code in a previously obtained frame and decodes it if found;
- Surface detection module: uses 4 points from the QR code and computes the perspective transform matrix;
- Content acquisition module: searches if required content is in the device's memory and downloads it if not found;
- Content transformation module: applies the perspective transform matrix on a content frame and obtains a frame that is ready to project;
- Content display module: projects the previously obtained frame on the camera preview.

In Figure 4.9 the content transformation module and content display module were treated as one step. As long as the content transformation module does not provide a new output, the content display module will always render the same image.

The client application stores all downloaded content in the device's memory so that it can be seen later on, without the need of contacting the server again. When the QR bar code is decoded and the content ID is extracted from it, the application must check if that specific content is available locally (in the device's internal storage). To be able to achieve that easily, we designed a fixed folder format. All content is stored with the following path: */sdcard/arinfo/[Language]/[ContentID]*, where Language and ContentID are replaced with the actual values. If the content file or language folder is not found, then a request to the server is made in order to download the missing content.

The content cache was implemented for a very important reason: loading content from

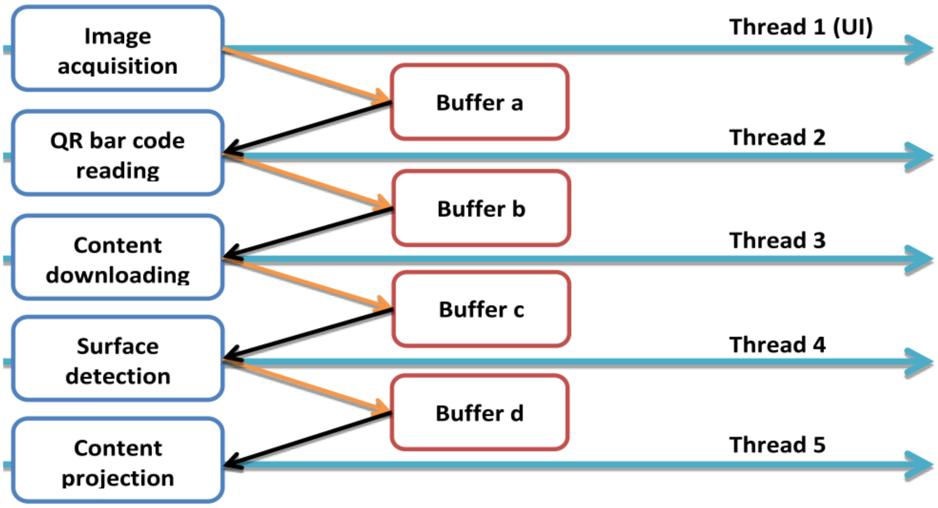


Figure 4.10: Buffer placement between threads

memory is a lot faster than downloading it from the server each time the user points the device to a QR bar code. In the case of images or animations the process is almost instantaneous whilst videos need more time to load. The cache was also needed for the buffering mechanism needed by accessing video files - the whole content cannot be loaded into memory (it can get quite big). A tradeoff between no cache at all and fast loading times would be using live video streams.

Persistent content storage is implemented on the server, using the Datastore and Blobstore provided by the Google App Engine API. The Blobstore is used to store the actual content files and a reference to it and all other metadata (such as language, ID etc) is stored in the Datastore. The server's interface allows an administrator to easily add content that can be obtained by the client.

In our implementation we used two Java libraries which are meant to assure that the application will work both correctly and fast: Open CV—for surface detection—and Zxing—for the reading of the QR codes. For the muted/soundless animations, we used the GIF format, from which we take individual frames but synchronize them with the main execution thread. Thus the speed is accurate, even though not all the frames are processed and displayed. In order to render audio/video materials, we take individual frames from their video source but we synchronize with the audio source. Unfortunately, the processing speed of this type of content drops considerably, so that the images end up being displayed with interruptions. It's worth mentioning that the porting of the ffmpeg library was necessary in order to display these videos.

The ffmpeg library was compiled using the Android Native Development Kit so that it can be used inside our Java application. The C code inside the library can be easily called from Java using the Java Native Interface. The library is mainly used to extract individual frames from the video file, at specific timestamps. The audio stream from the video file is rendered using the Android-specific MediaPlayer. The audio is synchronized with the video by using the same timestamps. The application's Content class offers uniform access to all file types that can be displayed. Thus, whether we deal with an image, animation or a movie, it insures that the content transformation module uses the right image. For videos or animations, the main execution thread is responsible with calling the nextFrame() method which signals to the Content class that a new image is needed.

The user interface consists of two activities:

- The Main Activity is used to display the camera preview as well as drawing the projected images. Important information coming from the operating system such as events (when the application is sent to background or foreground) is also received here. When the application starts or it is sent to foreground important operations are executed, such as starting the worker threads or obtaining access and initializing the device's camera.
- The Settings Activity is responsible with displaying the configuration options. The user may select the language of the displayed content, camera resolution or he may choose to turn on debugging information. Thus, the user can lower the resolution on low performance devices, so that the speed with which the content is displayed is increased.

The application relies on the images provided by the camera and it is not possible to run the application on a device that does not have a camera. The quality of the pictures provided by the camera does not matter as the application adapts to the resolutions it can provide.

Testing and Performance Evaluation

After the initial testing on the serial version of the application, we decided that using a threaded approach was mandatory. From 3 frames per second using the serial version, we reached more than 15 frames per second, when the same image was displayed. Functional tests were conducted on the threaded version in various conditions: using several devices, various types of content (images, animations, movies), different screen resolutions, etc. Figure 4.8 illustrates how content is seen on a device while debugging mode is enabled (frame rate and camera resolution is displayed). In this case, an animated content, matching the history museum scenario, was encoded in the QR bar code.

We also evaluate the performance, expressed as frames per second (*FPS*) in various conditions. Figure 4.11 shows the results of a test that aims to compare performance at different stages, with and without a QR code in view. We can see the number of frames processed every second for the important modules (Image acquisition, QR bar code reader and Content display), as well as the impact when a bar code was put in front of the camera and the application had to process it.

The test was done on a dual-core device running at 1.7 GHz. It consisted in a 40 seconds application run with no bar code visible for 12 seconds after which a marker pointing to a simple image was introduced to the device's view. The test showed that while no bar code is visible (application is mostly idle), the worker threads do not use as much power as they use while working. The idle state is characterized by the much higher frame rate achieved with no bar code visible. In Table II we can see the results of testing the performance of the application with three supported content types: images, animations and video. We pick a representative format for each of the content types.

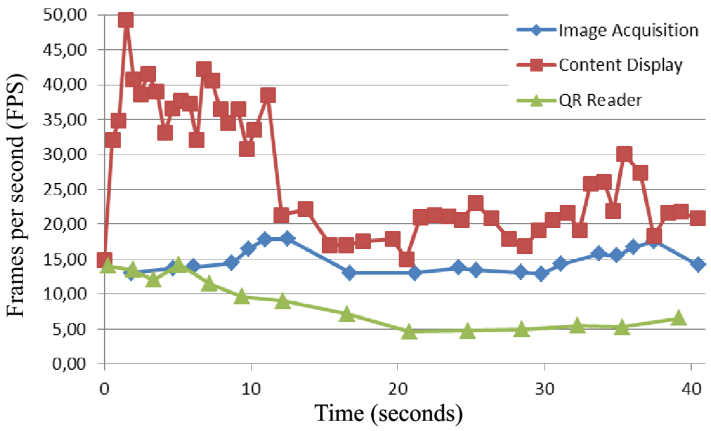


Figure 4.11: The number of frames processed by each module during an experiment

In Table 4.5 we summarize the results we obtain on four different devices, with different hardware characteristics. The FPS corresponds to the Content Projection Module, that is the final module in the processing loop, which is the speed the user is presented with new transformed images.

Table 4.5: Frame rate depending on device (left) and content (right)

Device	Cores	Resolution	FPS			Content Type	FPS
			Low	Medium	High		
SE Xperia	1 core @1GHz	320x480	23	18	13	Image (png)	23-27
Samsung GS2	2 cores @1.2GHz	540x960	34	29	24	Animation (gif)	20-27
HTC One S	2 cores @1.7GHz	480x800	40	35	30	Video (mp4)	15-20
Acer A510	4 cores @1.3GHz	1280x800	17	15	13		

Also in Table 4.5 we present the performance obtained for various types of projected content. Animation handling is very fast and it reduces to manipulating a series of simple images, as it can reach similar performance to image rendering. However, rendering video takes a lot of processing power because we use the ffmpeg library to extract individual frames at specific timestamps, which adds a lot of overhead. When an image needs to be projected (the display module is ready to draw another picture) the video needs to be seeked from the last position to the new timestamp.

Conclusion

We propose a client-server system, which aims to be a general purpose augmented reality application, which can be used in any scenario where rich and dynamic content can be overlaid over a QR code. We use surface detection to project the images, so that they seem to belong to the surrounding environment. As a client we develop an Android application and we implement the server using the Google App Engine platform to take advantage of its replication capabilities, based on demand from clients.

The client's architecture is based on a pipeline structure with the purpose of minimizing the delays of the module responsible with displaying images on the device's screen. Each module runs on its own thread, picks up input data from the preceding buffer and writes output data in the following buffer. This matches very well the design patterns of Android applications, where the main UI thread, responsible with displaying the camera preview, cannot be delayed.

Performance evaluation shows the amounts in which image processing speed is directly proportional with processor power and inversely proportional with the screen resolution (due to larger requirements in processing) and with camera resolution (as it takes more time to find the QR code).

4.2.2 Maintaining Performance by Offloading Stages in Processing Loops

As an alternative to the computation adaptation mechanism presented in Section 4.2.1, in this section we investigate offloading on OpenTTD, a popular open-source simulation game (Figure 4.12). In multiplayer mode, OpenTTD currently supports up to 255 simultaneous users on the same map, one of which must host the server. Another popular way of using the game is in singleplayer, against a number of AIs. The AIs are developed by the community and are freely available to all players.

Figure 4.12: Experiment running OpenTTD distributed on a tablet and laptop

System Design and Implementation

OpenTTD is a loop-based application, as it functions by continuously iterating over a processing loop, depicted in Figure 4.13. This is consistent with both the AR application presented in Section 4.2.1 and with the State Transition Diagram of our workload model presented in Section 3.3.4.

Essentially, the loop can be modeled as a cyclical graph, consisting of:

- input collection, either from the user or from an AI
- sending the local commands and receiving all-player commands from the server (only in networked games)
- simulation, which consists of several iterations on tiles, vehicles and building

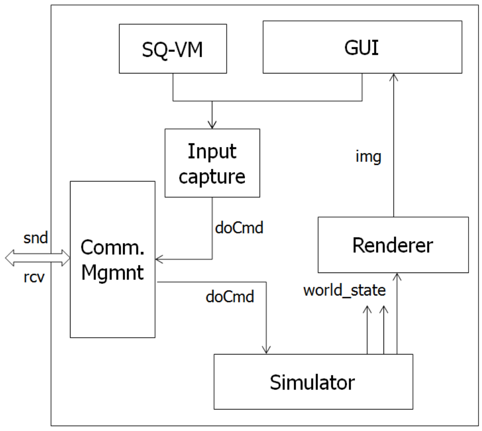


Figure 4.13: OpenTTD game loop

- rendering, which is greatly affected by the user graphics settings

For this experiment we choose to offload the AIs. Usually, the AIs are computationally intensive, as they often perform path-finding algorithms like A*, and bursty, as they only act when they have the in-game money to do so.

To enable the community to build custom AIs, the creators of OpenTTD designed them as scripts running in a Squirrel Virtual Machine, thus providing a loosely coupled interaction of the AIs with the whole system. This type of interaction also makes them fairly easy to offload.

For the implementation, we start adapting the source code of OpenTTD version 1.3.0, released in April 2013, one of the latest versions of OpenTTD. We implement offloading by tweaking a multiplayer game: the offloaded version is essentially a multiplayer game in which the client runs on the mobile device and the server runs the AIs. To do so, we make several changes to the community version.

We modify the game so that AIs can run in multiplayer, as this feature is normally disabled because the community doe not want to have players create AIs that compete instead of them in real matches. This modification is based on the adaptation made by Otto Visser from Delft University of Technology. We also implement instrumentation, to collect metrics both from the operating system level (like CPU load, memory load and network load) and from the application level (like frames per second and in-game time).

To conduct experiments, we need to repeat them for a number of times and make sure they all behave the same way. Thus we replace the human player with an AI running on the device and our implementation follows its movements on the screen. So, starting a game with the same AI on the same scenario will recreate in each run the same usage patterns and the same operations.

Finally, we adapt the starting procedure so that we can easily add more configuration settings in the original configuration file. This is necessary because when running Android applications it is not possible to start them with command line arguments.

To experiment with real mobile devices, we use the SDL port for Android written by Pelya. SDL is essentially the graphics library used by OpenTTD. Porting SDL for Android, the developer paved the way for porting any of the games using it. Thus, we are able to use

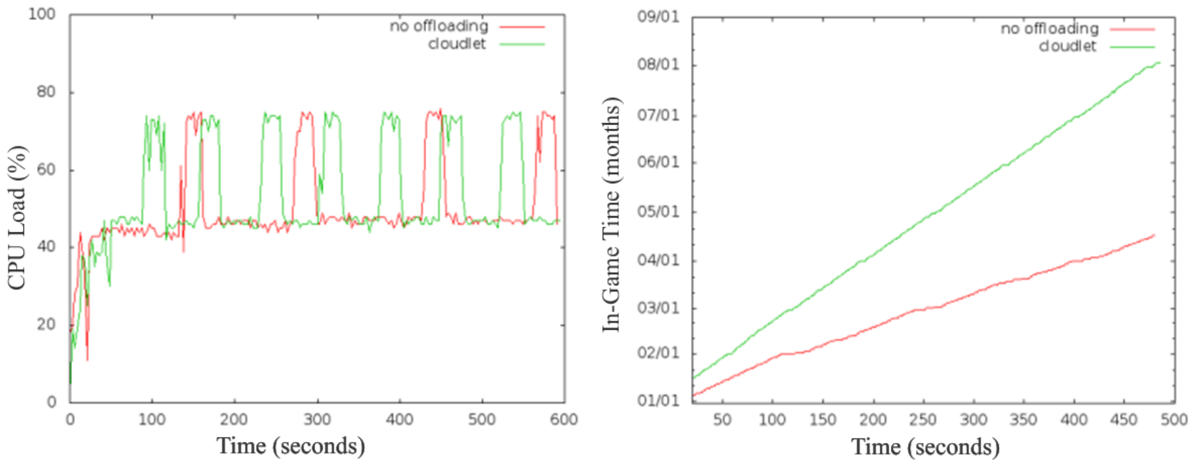


Figure 4.14: Comparison of local running of AIs (red) with offloaded running of AIs (green)

our modified C++ code, which is cross-compiled with the Android NDK and then linked with the rest of the SDL Android project.

Testing and Performance Evaluation

For our experiments we use, as client, an Asus Eee Pad Transformer TF101, featuring a nVidia Tegra 2 1GHz dual-core CPU, running Android 4.0.3. We also use, as server, a Sony Vaio laptop, with a Intel Core i3 2.27 GHz quad-core CPU, running Linux Ubuntu 10.04. The two devices connect through WiFi and are in the same Local Area Network.

We have tested the application with various game settings, such as details on or off, animations on or off, quality of textures high or low, maps large or small, etc. We present here an experiment using a large and dense map, and high graphics settings and we discuss more about calibration and the impact of game settings in Section 5.2.

The scenario we investigate is that a human player has a match against a number of AIs. We have started with 16 AIs, but running the game locally with that many AIs proves to be unplayable on our device. So, to collect data for a control test run, we repeatedly decreased the number of AIs until the game became usable on our device. We have settled to 4 AIs, that we picked from the most popular in the community, namely OtviAI, AIAI, ChooChoo and Chopper.

Figure 4.14 shows CPU load and in-game time we obtained during 10 minute gaming session, using our version of OpenTTD enhanced for repeatability.

The left-hand chart shows a CPU load of 40-50% most of the time, which corresponds to a high usage of one of the two cores. The spikes are triggered by autosaves, which, in our setting, take place once per in-game month. The autosaves are performed on a separate thread, which explains why the CPU load exceeds the 50% threshold. The right-hand chart shows how in-game time progresses related to real time. The vertical axis depicts the in-game time in months, while the horizontal axis depicts real time in seconds.

Both charts indicate that, without offloading, the game slows down to compensate for the lack of processing power of the client device. As indicated by the values in the right-hand chart, as well as by the number of spikes in the left-hand chart, in a 10 minute gaming session, the offloaded version covers almost 8 in-game months, while the local version covers only 4 in-game months.

Conclusion

We continue the analysis of loop-based applications we started in Section 4.2.1 by experimenting with computational offloading on OpenTTD, a popular open-source game. We use the multiplayer version of the game to experiment with offloading one of the stages in the processing loop, interpreting the AI player scripts. We modify the community version of the game so that it is suitable for experiments: running the game multiple times with the same parameters will produce the same usage patterns

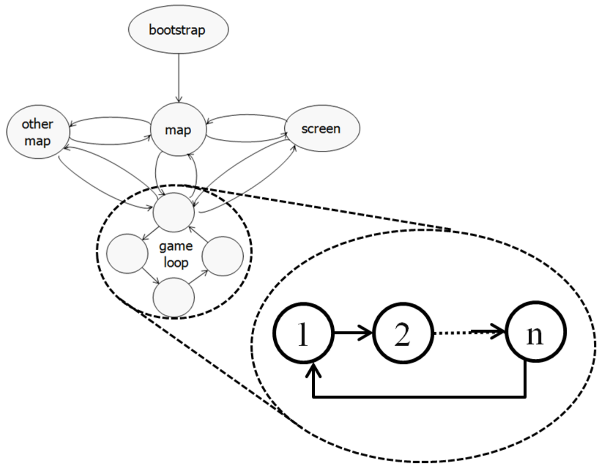
We find that, when running 4AIs locally the in-game action is twice as slow as it is when offloading the AIs. In some cases, performance adaptation is a reasonable solution to the lack of computing power. However, for highly interactive applications, such as games, slowing down the game speed is not a viable solution. In this case adaptation can lead to poor user experience, longer playing times for the same incentive, and, of course, more resources used by the client device.

4.3 Operational Analysis for Offloading in Loop-Based Applications

We continue our investigation on loop-based applications that we started in Section 4.2 and, in this section, we propose an operational analysis for loop-based applications, matching the offloading mechanisms we identified in Section 2.4.2 as part of our Exploratory Space.

As identified in Chapter 4, we focus on loop-based applications. A loop-based application is one in which most of the functionality is given by iterating an execution loop. All the Online Social Applications, for which we built a Workload Model in Chapter 3, have such a loop, as it can be seen in the Micro Model, in Section 3.4. In addition, for Online Social Applications, independent of their technology: Tightly Coupled Simulations, Web 2.0 or Streaming, some amount of functionality already occurs remotely, this being one of the criteria for which we chose them to conduct our studies.

We model the main loop of the application with a graph that consists of a cycle, with stages numbered from 1 to n, as depicted in Figure 4.15. An example of such a loop is given in Figure 4.13, which shows the main processing loop of OpenTTD that consists of stages such as input capture, synchronization on the server, simulation, and rendering. Based on this loop, and derived from the metrics and the benefit assessment models described in our taxonomy (Section 2.3) we express three main criteria for assessing the benefit of offloading, in terms of time (Equation 4.7), energy (Equation 4.8) and cost (Equation 4.9).



$$T = \sum_{i=1}^{n} T_{p|r}^{i} + \sum_{i=1}^{n} \frac{Q^i}{B(s,d)} \tag{4.7}$$

$$E = \sum_{i=1}^{n} E_{p|r}^{i} + \sum_{i=1}^{n} E_t(Q^i, B(s,d)) \tag{4.8}$$

$$C = \sum_{i=1}^{n} C_{p|r}^{i} + \sum_{i=1}^{n} C_t(Q^i, B(s,d)) \tag{4.9}$$

Figure 4.15: The main loop as a cycle graph.

where:

- T - time needed to perform computation or a data transfer
- E - energy consumed by computation or a data transfer
- C - monetary cost to be paid for a computation or a data transfer
- p/r - local/remote processing
- Q - quantity of data to be transferred
- B(s,d) - bandwidth when transferring data from source s to destination d

In the remainder of this section we make some simplifications, to keep the formulas concise, at the expense of some precision. First, we consider the download speed and the upload speed roughly the same size:

$$B(local, remote) = B(remote, local) = B \tag{4.10}$$

We also consider that passing data among local stages, as well as passing data among remote stages, is much quicker than passing data from local to remote and vice versa, and, therefore, we consider it as infinite:

$$B(local, local) \to \infty, B(remote, remote) \to \infty \tag{4.11}$$

Finally, when estimating energy, the energy consumed by remote resources can be discarded from the perspective of the client device. Similarly, we do not take into consideration any monetary costs for performing computations locally on the device.

$$E_r \approx 0, C_p \approx 0 \tag{4.12}$$

In the following, we expand our analysis based on the exploratory space of offloading mechanisms that we proposed in Section 2.4.2. For the first direction in the exploratory space, the number of components that are offloaded, we detail all three criteria—performance, energy and cost—and show the operational analysis is analogous. For the other four directions in the exploratory space, partial offloaded process, partial offloaded data, parallel offloading and resource placement, we only show the operational analysis for performance.

4.3.1 Offloading Various Components

As previously shown, we address loop-based applications and we model their processing with a cyclic graph. We are also focused on applications that already have some form of communication. We depict this by having a stage being processed in the environment (see Figure 4.16 a), as, for example, OpenTTD has the synchronization stage being processed on the server.

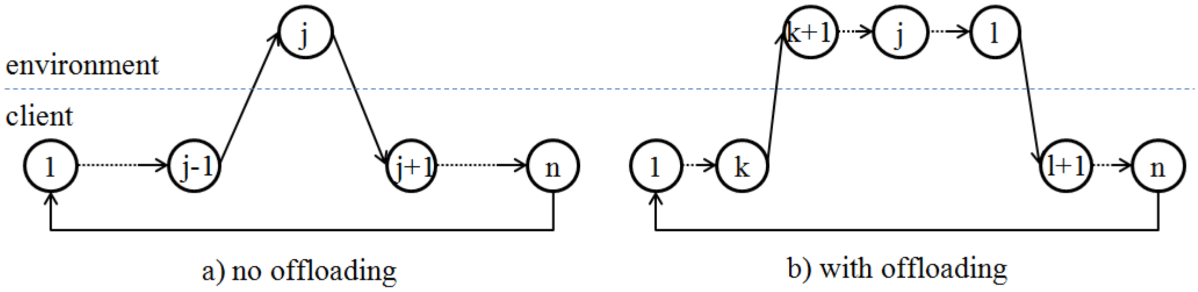


Figure 4.16: The graph model for varying offloaded components.

By offloading components we refer to moving the processing of additional stages in the environment. For example, in OpenTTD, besides doing the synchronization, we might remote the execution of the simulation as well. We generally note with k the stage where local processing turns into remote processing and with l the stage where processing returns on the device. Offloading is thus represented by moving indexes k and l within the loop, as described by Figure 4.16 b. We use this formalism to express the offloading decision logic, for the three main benefit assessment models: performance, energy and cost.

Inspired by [28] and [3], we express performance as the time it takes to perform the loop:

$$T(k,l) = \sum_{i=1}^{k} T_p^i + T_t^k + \sum_{i=k+1}^{l} T_r^i + T_t^l + \sum_{i=l+1}^{n} T_p^i \tag{4.13}$$

where T_p^i is the local processing time of stage i, T_r^i is the remote processing time of stage i, and T_t^i is the transferring time of output data from stage i. Transmission time can be expressed as the quantity of data transmitted divided by the transmission speed. At this stage we make the simplification that the sending and receiving speeds are similar. Local and remote processing times can be measured directly, and can also be expressed as the quantity of code to be executed divided by the CPU speed. The equation becomes:

$$T(k,l) = \sum_{i=1}^{k} T_p^i + \sum_{i=k+1}^{l} T_r^i + \sum_{i=l+1}^{n} T_p^i + \frac{Q^k + Q^l}{B} \tag{4.14}$$

In general, offloading is beneficial if remote processing has a better performance than local processing, or, equivalently, if the penalty for transmitting the data to the remote resource is less than gain in time obtained from using a remote resource more capable than the local one, expressed by the inequality:

$$\frac{Q^k + Q^l}{B} < \sum_{i=k+1}^{l} (T_p^i - T_r^i) \tag{4.15}$$

Thus, the offload decision becomes the optimization problem expressed as:

$$T_{min} = \min_{k,l} \left\{ \sum_{i=1}^{k} T_p^i + \sum_{i=k+1}^{l} T_r^i + \sum_{i=l+1}^{n} T_p^i + \frac{Q^k + Q^l}{B} \right\}, 1 < k < j < l < n \tag{4.16}$$

We can express the logic for energy and cost criteria in a similar way. For energy, we are interested in minimizing the energy consumed by the mobile device. Therefore, offloading is beneficial if the penalty for transmitting the data to the remote resource is less than the energy consumed by handling the stage locally. We can express this analogous to 4.14, with:

$$E_t(Q^k + Q^l) < \sum_{i=k+1}^{l} E_p^i \tag{4.17}$$

To explicit the transmission energy and the processing energy, we could use any of the energy models proposed in the literature, like [59] and [64]. These energy models are highly configurable and can cover any number of devices. Moreover, energy consumption can be measured directly with some limitations. In any case, the purpose of the offloading decision is the optimization problem expressed as:

$$E_{min} = \min_{k,l} \left\{ \sum_{i=1}^{k} E_p^i + \sum_{i=l+1}^{n} E_p^i + E_t(Q^k + Q^l, B) \right\}, 1 < k < j < l < n \tag{4.18}$$

For the cost criteria, we consider that communication and remote processing come with a cost. The application has to perform some stage remotely by its nature—let that be stage j— so it cannot function only locally. Therefore, assessing offloading benefit needs to compare offloading stages from k to l with executing stage j remotely:

$$C_t(Q^k + Q^l) + \sum_{i=k+1}^{l} C_r^i < C_t(Q^{j-1} + Q^j) + C_r^j, 1 < k < j < l < n \tag{4.19}$$

In this case as well, we can explicit the transmission cost and the remote processing cost, using various estimations. Therefore, the optimization problem for the cost criteria is:

$$C_{min} = \min_{k,l} \left\{ \sum_{i=k+1}^{l} C_r^i + C_t(Q^k + Q^l, B) \right\}, 1 < k < j < l < n \tag{4.20}$$

4.3.2 Intermittent Offloading

We refer to Intermittent Offloading in loop-based applications as offloading a component only once in a given number of iterations. We depict this using probabilities, matching our approach in modeling workloads at the operations level segmented on the State Transition Diagram (see Section 3.3.4. For example, processing advances from stage j to stage $j+1$ either moving to the device with probability $P(l)$ or staying on the remote resource with probability $1 - P(l)$:

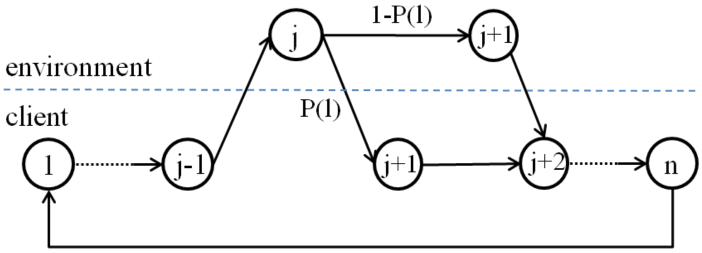


Figure 4.17: The graph model for partial offloading process.

Thus, the optimization problem becomes an issue of finding the probability $P(l)$ that minimizes the total iteration time. A similar formula can be derived for energy or cost. A weighted average of the three criteria can also be considered, for additional dimensions of the trade-off.

$$\begin{aligned} T_{min} = \min_{P(l)} \Bigg\{ & \sum_{i=1}^{j-1} T_p^i + T_r^j + P(l)T_p^{j+1} + (1 - P(l))T_r^{j+1} + \\ & + \sum_{i=j+2}^{n} T_p^i + \frac{Q^{j-1} + P(l)Q^j + (1 - P(l))Q^{j+1}}{B} \Bigg\} \end{aligned} \tag{4.21}$$

4.3.3 Offloading with Partial Data

When offloading part of the data, the same process, corresponding to at least one stage, is performed in the same time on the device and in the environment. For example, in OpenTTD, half of the map can be processed locally and half of the map can be processed remotely. A more suitable notion is the region of interest, which represents the part of the output that benefits from the user's attention, e.g. the viewport in games, that is the region of the map that the user sees at a given time. A finer grained notion is the object of interest. In any case, we represent the offloaded data as being the fraction R of the total data that is transferred from stage j to stage $j+1$.

The optimization problem is to find the ratio R for which, depending on the benefit model of choice, the iteration time, the energy consumption, or the monetary cost are at

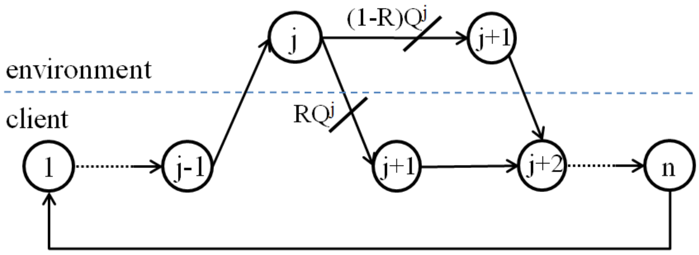


Figure 4.18: The graph model for partial offloading data.

a minimum.

$$T_{min} = \min_{R} \left\{ \sum_{i=1}^{j-1} T_p^i + T_r^j + \frac{T_p^{j+1}}{R} + \frac{T_r^{j+1}}{1-R} + \sum_{i=j+2}^{n} T_p^i + \frac{1}{B}(Q^{j-1} + RQ^j + (1-R)Q^{j+1}) \right\} \tag{4.22}$$

4.3.4 Parallel Offloading

Offloading can be done in parallel, that is using multiple remote resources in the same time for remote processing. For example, in OpenTTD a user can play against a number of AIs. We could offload AIs in parallel, that is each AI is processed by a separate remote resource. The trade-off is that dividing jobs properly in more tasks, provisioning sufficient resources and scheduling the tasks on the resources may be more consuming than the benefits of parallel processing. However, offloading itself, through its distributed nature, covers to some degree the division of tasks, provisioning and scheduling, which simply need to be tuned for this type of offloading.

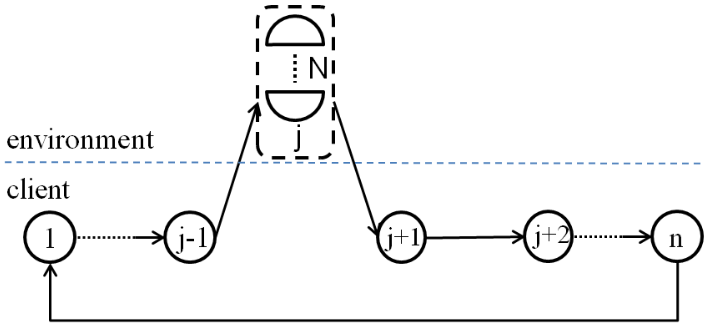


Figure 4.19: The graph model for offloading parallelism.

A key aspect when evaluating the benefits of parallel offloading is the amount of processing that cannot be parallelized, the serial part of the program S which we take into consideration by applying Amdahl's Law for the processing time of stage j remotely, we obtain:

$$T_{r||}^j = T_r^j(S + \frac{1-S}{N}) \tag{4.23}$$

We use this form of Amdahl's Law in Equation 4.7, but also consider that using N remote resources instead of 1 also means N times more communication. The performance optimization problem becomes:

$$T_{min} = \min_N \left\{ \sum_{i=1}^{j-1} T_p^i + T_r^j (S + \frac{1-S}{N}) + \sum_{i=j+1}^{n} T_p^i + N \frac{Q^{j-1} + Q^j}{B} \right\} \tag{4.24}$$

As the formula shows, using offloading parallelism might be beneficial when offloading in applications with a small serial part S and when each of the N remote resources can work with only part of the outputs from the previous stage.

Chapter 5

Performance Evaluation for Offloading Mechanisms

With inputs from all the other chapters of the thesis, in this chapter, we conduct experimental and analytical evaluation of several offloading mechanisms. We select our offloading mechanisms from the Exploratory Space defined in Chapter 2. For the empirical evaluation, we expand on our experiments on OpenTTD presented in Chapter 4 and for the analytical evaluation we use the operational analysis also described in Chapter 4, as well as the workload traces presented in Chapter 3.

5.1 Measuring Performance on Mobiles

Analyzing power consumption is an interesting research topic, as shown by various projects that can be found in the literature. Caroll and Heiser [59] design multiple micro-benchmarks to associate the power costs to modules of a mobile system. They try to determine the power consumption of different parts like CPU, GSM and WiFi. Zhang et al. [25] describe a power module construction technique that they use to characterize 3G, GSM, WiFi, CPU and screen, and to introduce PowerTutor, an Android application that can use these models for power consumption estimation on any device. We use such tools to estimate the power consumption of different components, but we found little solutions on estimating power consumption from the Bluetooth radio.

Balasubramanian et al. [64] show that 3G, GSM and WiFi incur a high tail energy overhead. Pering et al. [68] describe methods to reduce the power consumption by switching between Bluetooth and WiFi. The Bluetooth module has a power consumption as much as 10 times lower than WiFi. WiFi is intended for high-bandwidth and 100 meters coverage while Bluetooth is designed for low-bandwidth and a coverage of 10 meters. The authors also describe that power consumption in idle mode compared to active mode is 4 times smaller for WiFi and 6 to 10 times smaller for Bluetooth. We find this significant as it justifies the need of advanced algorithms that power down these interfaces when communication is not necessary.

Various other papers try to determine algorithms and technologies for reducing energy consumption, by modeling optimum power as a Nash equilibrium [69], by employing backoff methods for synchronization [70], or introducing an active-sleep duty cycle [71]. We propose an adaptive algorithm for reducing the polling rate in location-aware applications, leveraging the fact that new queries are not necessary if the location has not changed significantly.

Some research was conducted in bringing new communication technologies, like IEEE 802.15.4, in use on the smartphones [63][64]. Our work investigates this prospect specifically in the context of home automation and its particular challenges. We also investigate the use of multiple channels, such as WiFi to connect to a central server, and ZigBee, through an USB dongle, to connect to a network device.

The concern for usability appeared among the first patents for home automation [72]. With the advancements in human-computer interaction in modern technology, researchers envision systems based on gestures for home control [73]. Our work is oriented towards bringing user control to off-the-shelf devices, such as smartphones and tablets.

We are also aware of the energy consumption challenges and we try to investigate that. We use smartphone specific methods for energy estimation and we apply some of the principles described in the research literature [25].

In our work we apply techniques of Computer Systems Performance Analysis that are well established in the field, providing very useful tools in the form of analytical modeling. Kleinrock [74] provides a very thorough introduction in the modeling of stochastic systems of flow using queuing theory. By understanding the stochastic processes that describe the arriving stream and the structure and discipline of the service facility, one can mathematically obtain measures of performance and effectiveness. Menasce [75] also uses queuing networks to obtain descriptive models for various types of systems, which serve as basis for quantifying performance models. Jain [76] offers a thorough reference to fundamental and practical engineering principles of computer systems evaluation and King [77] extends the principles on communication topics.

5.2 Empirical Evaluation

In our empirical evaluation, we continue our work presented in Section 4.2.2, where we detail an offloading experiment on OpenTTD, a popular open-source game. We design and implement a testbed based on OpenTTD, which we use to conduct a design space exploration for offloading mechanisms, based on the Exploratory Space we proposed in Section 2.4.2. In the remainder of this section, we detail the experimental setup based on the testbed, and the results we obtain in the experiments for calibration, offloading various components, and parallel offloading.

5.2.1 Experimental Setup

To conduct empirical evaluation of various offloading mechanisms, we design and implement a testbed based on OpenTTD, a popular open-source game. OpenTTD is the open-source version of Transport Tycoon Deluxe, a business simulation game developed by Chris Sawyer in 1994. As an open-source application, OpenTTD has a community of developers that continuously update the game and publish all sources in a repository. Moreover, other developers port the game to other platforms. For example, the Android port by Pelya has hundreds of thousands of active users.

We select OpenTTD because it is a real popular application, which falls in the category of online social applications, implemented as a tightly coupled simulation, for which we have a workload model, as proposed in Chapter 3. Tightly Coupled Simulations are, among Online Social Applications, the type of applications that provide opportunity for the broadest range of experiments, because the largest amount of processing takes place on the mobile device. We conduct static and dynamic analysis on OpenTTD. We use Vprof to profile OpenTTD on a Linux system and Vtune on an Android system. The basic functionality of the game consists of iterating a loop, which means we can apply our operational analysis described in Section 4.3. We detail the functionality of the loop in Section 4.2.2.

In multiplayer mode, OpenTTD follows a client-server architecture that currently supports up to 255 simultaneous users on the same map, one of which must host the server. There are some notable efforts to expand that number of users with a Massively Multiplayer Online version of OpenTTD, named At Large. In our testbed, we augment the client-server architecture, by tapping into the normal data flow of the system and adding an observer that also has the ability to control the experiments (see Figure 5.1). The testbed enables experiments for conducting offloading from various clients to either a cloudlet device, in the same LAN, or with powerful cloud infrastructure, over the Internet.

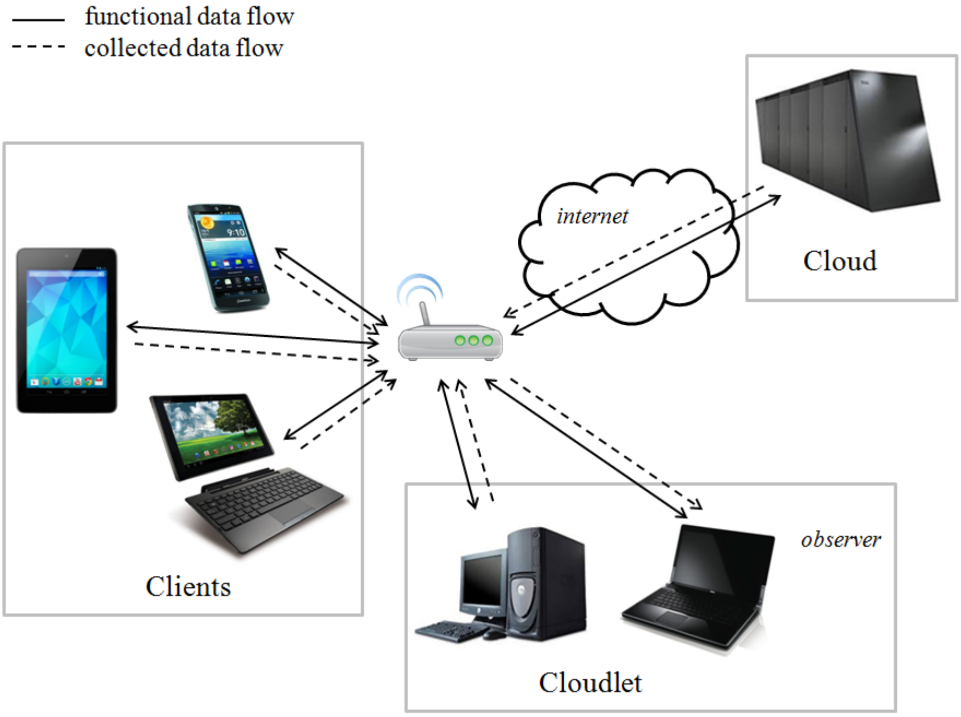


Figure 5.1: General architecture of the Testbed

We implement the testbed by making several changes to the community version. To conduct experiments, we need to repeat them for a number of times and make sure they all behave the same way. Thus we replace the human player with an AI running on the device and our implementation follows its operations on the screen. So, starting a game with the same AI on the same scenario will recreate in each run the same usage patterns and the same operations. To enable running AI players on the server, we adapt the code, starting with the modifications implemented by Otto Visser, from Delft University of Technology. In addition, we also adapt the starting procedure so that we can easily start the game on the mobile device and on the server through scripts. When starting Android applications, it is not possible to give the native code command line arguments, so it was necessary to implement some additional configuration settings in the original configuration file.

We also implement instrumentation in all the components of the architecture, to collect various metrics, as depicted in Figure 5.2:

- on the device, within OpenTTD: we measure game specific metrics, like frames per second and in-game time, as well as component statistics, in terms of processing time and quantity of data
- on the device, at application level: we run several profilers, such as vprof and VTune, and tools, like top and iftop for Android, to collect metrics such as CPU load, memory load, and network load
- on the device, in the kernel: hooks and system calls can be placed for forwarding to higher levels essential information captured from hardware counters on the physical device, like the C-states; we have investigated C-states for a better understanding on CPU loads [78], but we do not detail them in this testbed
- at the network level, software known as package sniffers, such as Fiddler and Wireshark, capture packets and help with statistics, such as sent and received packets, sent and received bytes, session length, inter-arrival times, and the size of the input and output data
- on the server side

We use top, iftop and the sysfs to monitor internal resources on the Android device. Network load is monitored using iftop and tcpdump. We use the tools AWS provide for the cloud servers and simple linux tools for our cloudlet machines.

The OpenTTD implementation already handles issues such as lagging clients and out of order commands. Each client holds a tick-based internal counter, that serves as a reference for the client in executing commands. A comprehensive error handling system can press the client to speed up on executing older commands, and can even go to the point of kicking out a client that cannot keep the pace.

We have created a suite of *bash* scripts that, when run from the observer, trigger the game on the clients and the server, start the measuring tools, wait for the time specified for the experiment and cleanly close all the programs that they started. All communication with the Android clients is done through *adb*, or Android Debug Bridge, a tool offered by Google, which enables file transfers and remote connections to the Android devices. Communication with the other devices in the cloudlet and the cloud is done through *SSH* and *SCP*, two popular communication protocols for remote connections and file transfers, with many clients on both Windows and Linux. At the end of the experiment, all the results are centralized on the observer, where other scripts automatically compute statistics and plot charts using *gnuplot*.

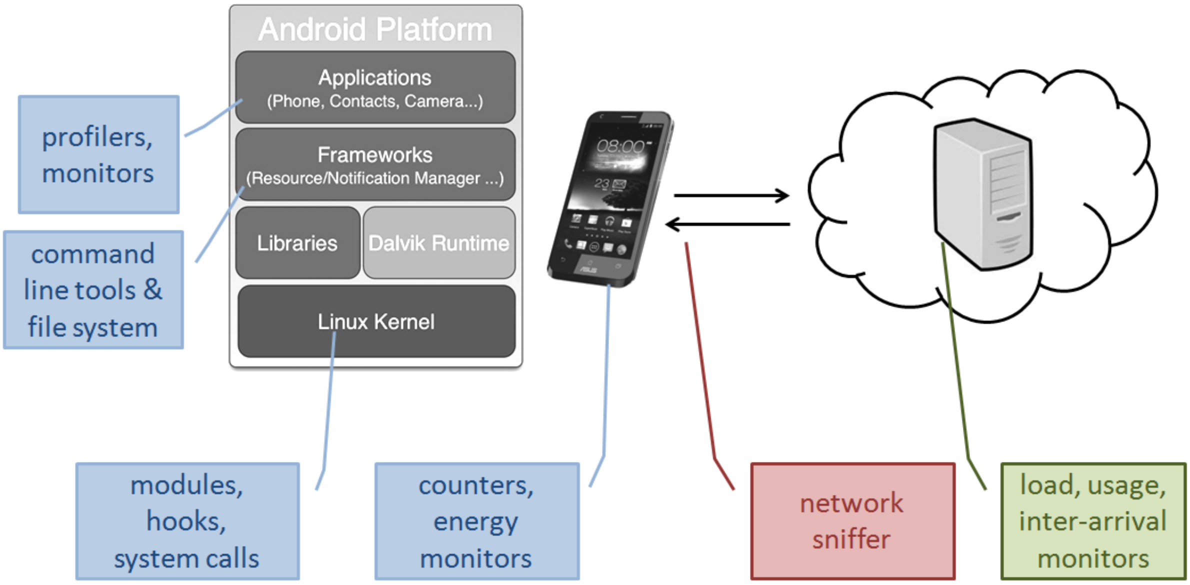

Android Platform
Applications
(Phone, Contacts, Camera...)
Frameworks
(Resource/Notification Manager ...)
Libraries
Dalvik Runtime
Linux Kernel
profilers, monitors
command line tools & file system
modules, hooks, system calls
counters, energy monitors
network sniffer
load, usage, inter-arrival monitors

Figure 5.2: Instrumentation in our Offloading System

The system does not take a dynamic decision, but is instructed by the observer which tests to run, because we are interested in comparing various mechanisms. We implement several variants of OpenTTD, to offload different amounts of components. We also allow file configuration for varying parameters, such as graphics and user behavior.

5.2.2 Empirical Results

In this subsection, we describe the analytical evaluation of two offloading mechanisms from the Exploratory Space, namely offloading component variation and parallel offloading.

However, before proceeding with the empirical evaluation of the two offloading mechanisms, we compare effects that various parameters have on performance, parameters such as graphics quality and map size.

Calibration: effects of parameters

In this first experiment, we want to evaluate the influence various game parameters have on our implementation. The purpose of this experiment is to let us select a configuration suitable for the other experiments. We investigate graphics level and scenario type as game parameters with great impact on the performance.

Graphics level usually has a great influence on the rendering performance. We define graphics as being high or low by collectively setting color-depth, animations and details. Color-depth denotes the number of bytes used to store color information for each pixel. In OpenTTD, graphics can be represented with 8-bits or 32-bits per pixel. Thus, the color-depth can lead to four times the size of the rendered images. Animations refer to the small movements various artifacts do on the screen, and require additional processing, as they are usually implemented by iterating over an animation loop. The level of details determines how many artifacts each tile has. Therefore, we define:

$$graphics : high = \{color : 32 - bit; animations : on; details : on\}$$
$$graphics : low = \{color : 8 - bit; animations : off; details : off\}$$

Scenario sizes usually have a great influence on the simulation stage, which usually needs to iterate through the whole world representation, but also on the AI input capture, because AI players need to traverse the whole world when taking decisions. We define scenarios of various sizes by selecting three predefined scenarios from the community:

$$Europe : large\&dense = \{1024x1024tiles;\ 10.000citiesandresources\}$$
$$Antarctica : large\&sparse = \{1024x1024tiles;\ 100resources\}$$
$$2Mountains : small\&dense = \{128x128tiles;\ 100resources\}$$

We try to use scenarios that resemble geographical units from the real world, because users generally tend to play on them.

For this experiment, we use our *openttd-repeatable* implementation, which allows us to repeat the same scenario multiple times. We run the client on an Asus TF101 2-core @1GHz running Android 4.0.3 and the server on a Sony Vaio 4-core @2.3GHz running Ubuntu 10.04, as described in the Experimental Setup 5.2.1. There is one AI player running on the client to simulate a real player.

We compare the time to complete an iteration on the client, when varying the graphics and, respectively, when varying the scenario. Figure 5.3 summarizes the full range of values that we obtain in the form of CDFs.

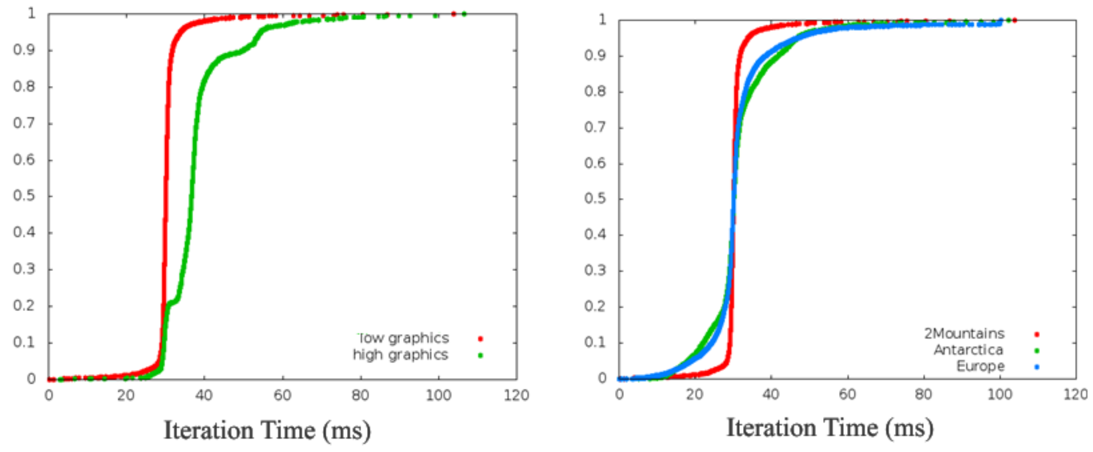


Figure 5.3: Calibration: effects of game parameters

It can be noted that using low graphics on a small map in 90% of the cases the iteration time is less than 30ms, while using high graphics only in 20% of the cases the iteration time is less than 30ms and in 90% of the cases it steeps to 60ms. Varying the map size also has a significant impact because in 90% of the cases the iteration time is less than

30ms using a small map, which covers only 70% of the iterations when using a large map. The map density does not seem to have a significant impact, as the difference between the Europe and Antarctica scenarios is fairly small.

Offloading Various Components

For this experiment, we explore the benefits of offloading the AI Input Collection component of the game. We represent the AI Input Collection as a stage in the processing loop of OpenTTD (see Section 4.2.2. Profiling shows that the AI players can consume significant amounts of processing power when computing routes from one city or resource to another, which they do by using algorithms such as A*, especially when the tree representation of the world is large. However, the AI players have a bursty behavior, and only compute such routes only when they acknowledge they have enough in-game money. It is therefore interesting to see what is the impact of offloading the Squirrel Virtual Machine, which runs the AI scripts and offers an AI input.

Using this experiment, we explore one of the four main offloading mechanisms identified in our Exploratory Space, as depicted in Figure 5.4. We compare not offloading anything, that is running AIs on the client device, with offloading the AI component, that is running the AIs on the server. We use our testbed and, as offloading target, the cloudlet represented by a Sony Vaio laptop.

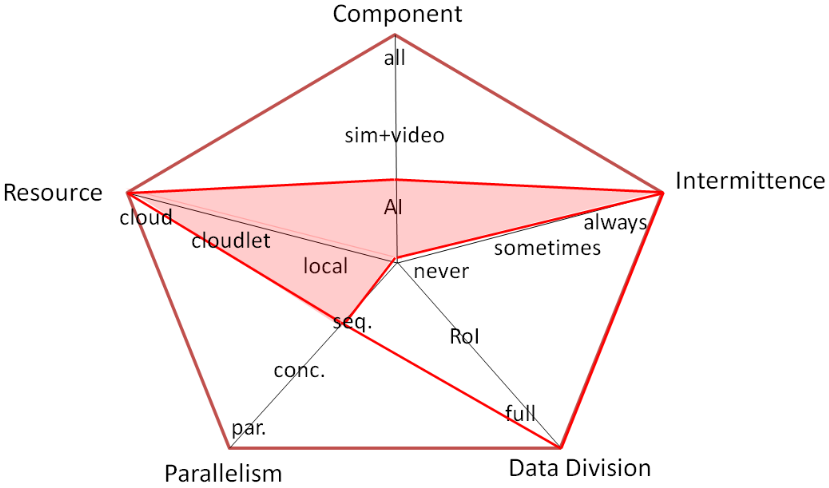
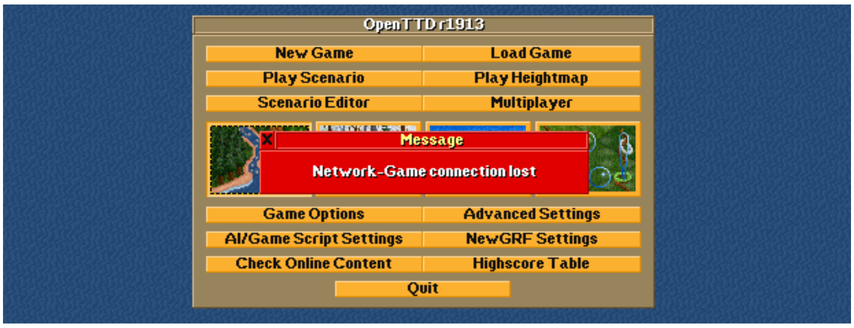


Figure 5.4: Exploratory Space Coverage for Variation of Offloaded Component

We conduct the experiment several times, increasing the number of AI players in the game. We find that our client device, an Asus Transformer TF101, has problems running the game with 4 AI players. The client is removed from the game (see Figure 5.5), because it fails to send commands to the server often enough. All clients must start a new loop every 100 ms to ensure fairness for all participants and send commands to the server each loop. In our measurements, the iteration time is generally between 100 ms and 200 ms because our measurements follow the data flow: commands sent in one iteration to the server will be sent back to the client in the next iteration, to hide latency.

Figure 5.6 reports two of the metrics we collect, showing the first 12 minutes of the experiment. The iteration time is a performance metric, and shows how long an iteration



Network-Game connection lost

Figure 5.5: Clients that do not perform offloading might be removed from the game because they are too slow

of the data takes to be processed through the whole loop. The in-game time is a metric closely related to user experience, and shows how much time passed in the game world related to real time. In-game time is important to the whole experience because, on one hand, if it passes too slow then the user feels the game lagging and, on the other hand, advances in the game are triggered by the passing of in-game time.

We log the iteration time in our modified version of OpenTTD every time an iteration completes. With roughly ten iterations per second, during a 10 minute window we would have roughly 6,000 readings. To present our findings in a friendly way, we aggregate the readings on a per-minute basis, and we only report the median value in the left chart in Figure 5.6. It can be seen how in the no-offloading version, the iteration time increases in the first 5-7 minutes of the game and gets above the threshold in the 8th minute, when the client is removed from the multiplayer. On the other hand, the median of the offloaded version is not affected much and stays all the time in the range of 160-165 ms.

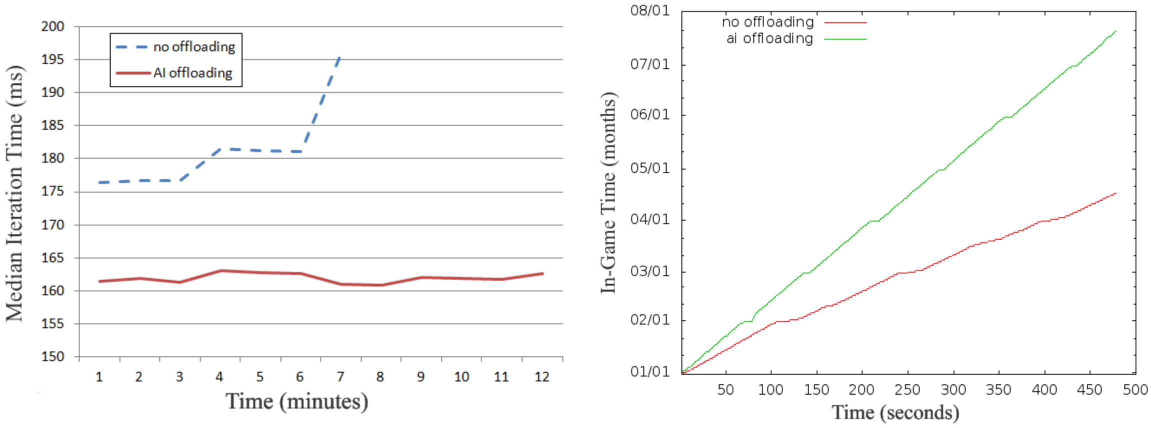


Figure 5.6: Performance and user experience metrics for Offloading Various Components

The in-game time shows an important difference (see the right chart in Figure 5.6), consistent with our findings in Section 4.2.2. In the first 470 seconds of the experiment, the no-offloading version advances from January 1st to April 4th, while the offloading version advances from January 1st to July 21st. This shows that the no-offloading version advances approximately twice as slow as the offloading version, the user will experience lags and will have to play twice the amount of real-world time in order to get the same results in the game.

Parallel Offloading

Parallelism is already exploited at some levels in OpenTTD. Profiling shows that tasks such as saving the game and playing the audio files are operated on separate threads. We have also seen that the clients send to the server commands from one iteration, but receive the commands from the previous iteration, to hide network latency.

In this section, we exploit that running multiple AIs to offer one move is an embarrassingly parallel decision. Each AI reads the game state and there is no communication and no race condition among the different AIs. Thus, we try to leverage the power of the computing infrastructure, by running AIs in parallel.

We repeat the experiment from the previous section where the mobile client is just a spectator and we run 4 AI players on the laptop representing our cloudlet. For comparison, we also run the 4 AI players as separate clients. The laptop has a quad-core CPU, which allows for the 4 AI players to be run in parallel. Thus, we explore another of the four main offloading mechanisms identified in our Exploratory Space, as depicted in Figure 5.7.

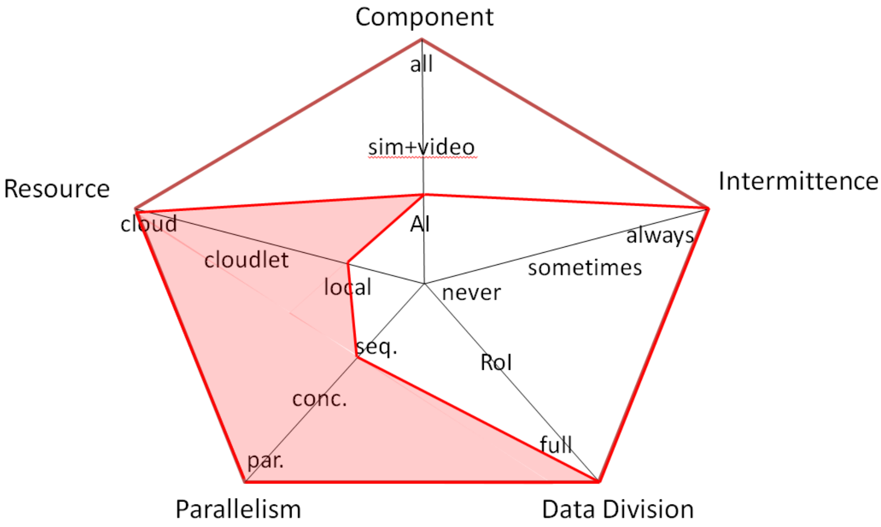


Figure 5.7: Exploratory Space Coverage for Parallel Offloading

We present results for the same two metrics as in the previous experiment and also for network traffic. The iteration time is a performance metric, and shows how long an iteration of the data takes to be processed through the whole loop. The in-game time is a metric closely related to user experience, and shows how much time passed in the game world related to real time. Network traffic is a penalty that the client must pay in order to benefit from the advantages of the more powerful remote infrastructure.

All clients must start a new loop every 100 ms to ensure fairness for all participants and send commands to the server each loop. So, in our report window of 10 minutes there are readings for approximately 6,000 loops. To present our findings in a friendly way, we aggregate the readings on a per-minute basis, and we only report the median value in the left chart in Figure 5.6. As the chart shows, there are no significant improvements, as the median iteration time for both variants stays in the range 160-165 ms. This is probably caused by the manner in which we split the AIs into parallel tasks, by using them in

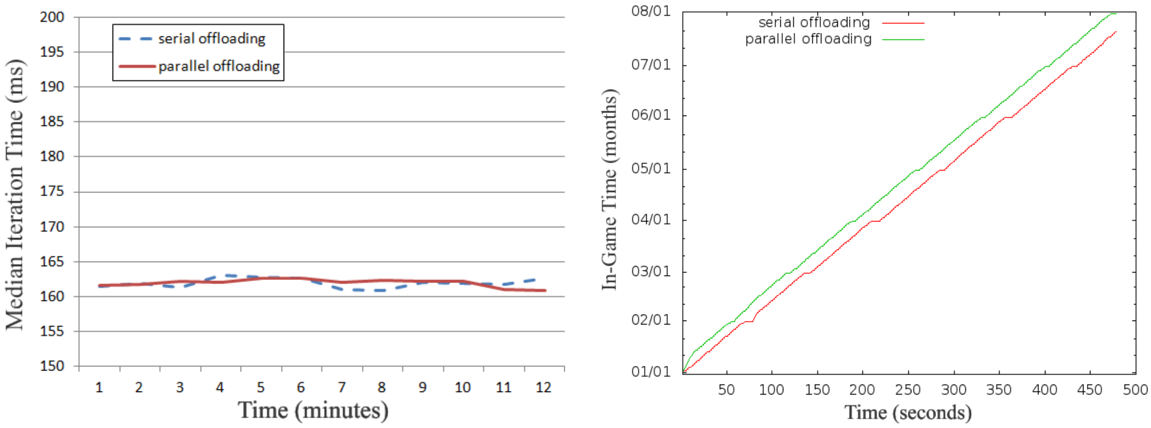


Figure 5.8: Performance and user experience metrics for Parallel Offloading

separate clients, which makes the parallel part insignificantly small in comparison with the serial part.

The in-game time shows a relevant difference (see the right chart in Figure 5.6). In the first 8 minutes of the experiment, the serial version advances from January 1st to July 21st, whereas the parallel version advances from January 1st to August 1st. Thus, the serial version advances 201 in-game days, with 11 days less than the parallel version, which accounts for a 5.47% speed-up on the parallel version.

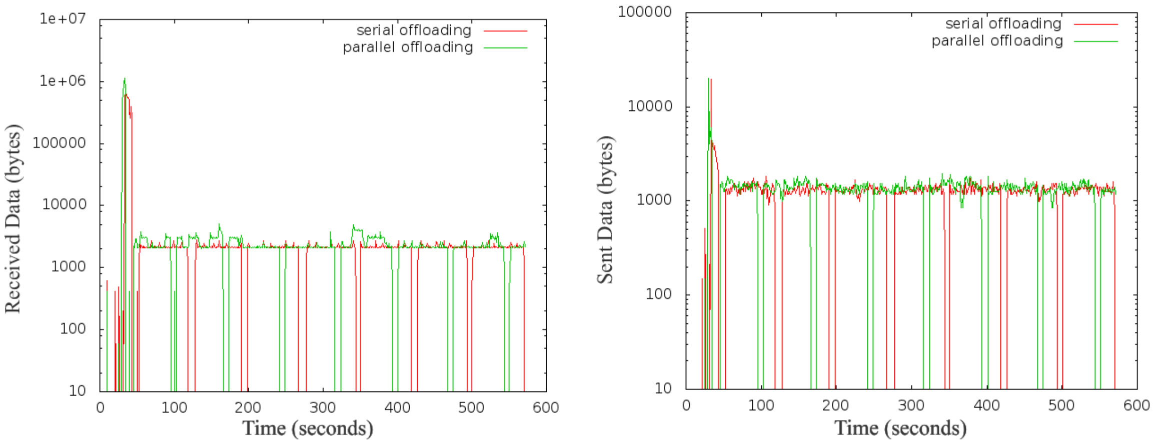


Figure 5.9: Network load for Parallel Offloading

We are also interested in observing the penalty in data transfers. Figure 5.9 shows, on a logarithmic scale, the amount of bytes sent by the two versions during the first 10 minutes of the experiment. As it can be seen, the data transfers are similar among the two versions, with a large spike during the initialization phase, when content is downloaded, followed by a leveled amount of data transferred in time.

In our case, we choose to offload all the data to a central component, that, in turn, transfers it to the workers executing the code remotely. This approach allows for a constant level of transfers and it can also explain the relatively low performance improvement.

5.3 Analytical Evaluation

In this section, we use the operational analysis described in Chapter 4 and the workload traces presented in Chapter 3 to evaluate a selection of two offloading mechanisms on OpenTTD, the same application we use in our empirical evaluation in Section 5.2.

5.3.1 Analysis Parameters

Based on the State Transition Diagram, an element of our Micro Workload Model (see Section 3.3.4), we model the loop of the application using a cyclical graph (Figure 4.16). For OpenTTD we consider five stages: human input collection, AI input collection, command synchronization, simulation and rendering. Thus, the graph will have $n = 5$ stages.

When deciding which components to offload, we should consider the portability of the stages:

- Stage 1, human input collection, can only be handled on the device
- Stage 3, command synchronization, needs to take place remotely, on the server, because it needs to take inputs from all the clients
- Stages 2, 4 and 5, AI input collection, simulation and rendering, can be handled on both device and server

We show in our characterization and modeling of OpenTTD traces (see Section 3.4.4) that processing time of Stage 2, the AI input collection, differs between a tablet and a laptop with one order of magnitude (on average 48.2 ms on the tablet and 4.3 ms on the laptop), while keeping a small size of output data (approximately 40 bytes). On the other hand, the time difference for simulation is not as large (on average 9.3 ms on the tablet instead of 2.6 ms on the laptop) and comes with the cost of transferring a larger size of output data (approximately 1MB). The same considerations apply for rendering, which may be beneficial if using streaming, but already has a comparable processing time on the device and the tablet. Thus, in the remainder of this section, we use Stage 2 to analyze two offloading mechanisms from our Exploratory space, component offloading and parallel offloading.

Another key parameter in our analysis is the bandwidth. The bandwidth can vary greatly from one implementation to another, influenced both by hardware and software stack. In our experiments, we use WiFi, which can offer a nominal speed of up to 54 Mbps. However, we our application may not be the only one using the wireless. On Android devices specifically, there are a number of background services that may perform synchronization while other applications are running. Based on our experience on the empirical evaluation, we estimate an average speed of $8Mbps$.

5.3.2 Analytical Results

In this subsection, we describe the analytical evaluation of two offloading mechanisms from the Exploratory Space, namely offloading component variation and parallel offloading.

Offloading Various Components

We compare offloading the AI input collection stage with no offloading, using Equation 4.14. We also summarize in Table 5.1 the values we use, based on the values from workload modeling (see Section 3.4.4).

Table 5.1: Data used in analytical evaluation.

Stage **name and number**	**Local** **Processing**	**Remote** **Processing**	**Quantity** **of Data**
	T_p [ms]	T_r [ms]	Q [byte]
Human Input Collection (1)	1.2	0.7	40
AI Input Collection (2)	48.2	4.3	40
Command Synchronisation (3)	188.5	101.9	120
Simulation (4)	9.3	2.6	1,068,576
Rendering (5)	5.7	13.7	4,096,000

Bandwidth
B [Mbps]
8

In the base case, without offloading, $k = j = l = 3$, indicating the remote operation of command synchronization, the only stage that needs to be performed remotely:

$$T(3,3) = T_p^1 + T_p^2 + T_r^3 + T_p^4 + T_p^5 + \frac{Q_2 + Q_3}{B} = 166.45ms \tag{5.1}$$

In a similar way, when offloading Stage 2, the AI input collection, $k = 2$, $j = l = 3$, and the formula becomes:

$$T(2,3) = T_p^1 + T_r^2 + T_r^3 + T_p^4 + T_p^5 + \frac{Q_1 + Q_3}{B} = 122.55ms \tag{5.2}$$

The smaller time for remote processing Stage 2 compared with local processing, as well as the comparable output data size of Stage 1 and Stage 2, make offloading Stage 2 a better option than no offloading.

In comparison, trying to offload Stage 4, the simulation would lead to:

$$T(3,4) = T_p^1 + T_p^2 + T_r^3 + T_r^4 + T_p^5 + \frac{Q_2 + Q_4}{B} = 1178.71ms \tag{5.3}$$

This would be an example when offloading is not beneficial. Due to the large size of the output data, one iteration of the loop would take more than a second, which means that the client device will be removed from the multiplayer game because it is too slow.

We now consider Stage 2' to describe running 4 AI players instead of 1. Having multiple AI players take a decision during one iteration is an embarrassingly parallel task, thus running N AI players on a serial processor would take N times the time. This, for $N = 4$ players, the base case can be represented as:

$$T(3,3) = T_p^1 + T_p^{2'} + T_r^3 + T_p^4 + T_p^5 + \frac{Q_2 + Q_3}{B} = 311.05ms \tag{5.4}$$

and offloading Stage 2' becomes:

$$T(2',3) = T_p^1 + T_r^{2'} + T_r^3 + T_p^4 + T_p^5 + \frac{Q_1 + Q_3}{B} = 135.45ms \tag{5.5}$$

Thus, when running 4 AI players locally, the 200 ms loop period is exceeded, and the client will be removed from the multiplayer game. In this case, offloading is necessary, as it brings the iteration time below the 200 ms threshold.

Intermittent Offloading

We have seen how offloading $N = 4$ AI players can have significant benefits on performance. We would like to see what benefits parallel offloading can have on this embarrassingly parallel task. Applying Amdahl's law, we assume that the serial portion of this code $S = 0$ and thus, applying Equation 4.23 we obtain that $T^j_{r||} = \frac{T^j_r}{4}$).

Thus, the processing time of one iteration becomes:

$$T(2'_{||},3) = T_p^1 + \frac{T_r^{2'}}{4} + T_r^3 + T_p^4 + T_p^5 + 4\frac{Q_1 + Q_3}{B} = 119.78ms \tag{5.6}$$

We compute the relative benefit RB of offloading in parallel using Equations 5.4, 5.5 and 5.10 as:

$$RB = \frac{T(3,3) - T(2',3)}{T(2',3) - T(2'_{||},3)} = 8.9\% \tag{5.7}$$

In this case, offloading in parallel brings an additional benefit of 8.9%. The relative benefit can serve as an easy indicator for server operators who want to implement several offloading mechanisms, to decide whether additional implementations are worth the costs.

Offloading with Partial Data

We have seen how offloading $N = 4$ AI players can have significant benefits on performance. We would like to see what benefits parallel offloading can have on this embarrassingly parallel task. Applying Amdahl's law, we assume that the serial portion of this code $S = 0$ and thus, applying Equation 4.23 we obtain that $T^j_{r||} = \frac{T^j_r}{4}$).

Thus, the processing time of one iteration becomes:

$$T(2'_{||},3) = T_p^1 + \frac{T_r^{2'}}{4} + T_r^3 + T_p^4 + T_p^5 + 4\frac{Q_1 + Q_3}{B} = 119.78ms \tag{5.8}$$

We compute the relative benefit RB of offloading in parallel using Equations 5.4, 5.5 and 5.10 as:

$$RB = \frac{T(3,3) - T(2',3)}{T(2',3) - T(2'_{||},3)} = 8.9\% \tag{5.9}$$

In this case, offloading in parallel brings an additional benefit of 8.9%. The relative benefit can serve as an easy indicator for server operators who want to implement several offloading mechanisms, to decide whether additional implementations are worth the costs.

Parallel Offloading

We have seen how offloading $N = 4$ AI players can have significant benefits on performance. We would like to see what benefits parallel offloading can have on this embarrassingly parallel task. Applying Amdahl's law, we assume that the serial portion of this code $S = 0$ and thus, applying Equation 4.23 we obtain that $T_{r||}^{j} = \frac{T_r^j}{4}$).

Thus, the processing time of one iteration becomes:

$$T(2'_{||}, 3) = T_p^1 + \frac{T_r^{2'}}{4} + T_r^3 + T_p^4 + T_p^5 + 4\frac{Q_1 + Q_3}{B} = 119.78ms \tag{5.10}$$

We compute the relative benefit RB of offloading in parallel using Equations 5.4, 5.5 and 5.10 as:

$$RB = \frac{T(3,3) - T(2',3)}{T(2',3) - T(2'_{||},3)} = 8.9\% \tag{5.11}$$

In this case, offloading in parallel brings an additional benefit of 8.9%. The relative benefit can serve as an easy indicator for server operators who want to implement several offloading mechanisms, to decide whether additional implementations are worth the costs.

5.4 Comparison

In this section we compare the results we obtained during our experiments with the ones obtained by applying the operational analysis, to validate our understanding on the two offloading mechanisms: component variation offloading and parallel offloading. In our operational analysis, we refer to a couple of performance metrics: *total iteration time* and *quantity of data transmitted over the network*. Table 5.2 summarizes the significant figures in our comparison. We consider two base cases and two experiments:

- *Base Case 1: No network* - we run OpenTTD in singleplayer mode on the mobile client; no communication is performed by OpenTTD, but it still has to run 4 AI players
- *Base Case 2: No offloading* - we run OpenTTD in multiplayer mode, on the mobile client and the laptop; now the client needs to send commands to the server for synchronization, but nothing else is offloaded
- *Experiment 1: AI serial offloading* - we run OpenTTD in multiplayer mode, on the mobile client, and on the laptop several clients run an AI player each, but they are all forced to work on a single core, so they operate in serial
- *Experiment 1: AI parallel offloading* - we run OpenTTD in multiplayer mode, on the mobile client, and on the laptop several clients run an AI player each, in parallel

For iteration time, we compute the average in the four cases—no network, no offloading, offloading the AIs in a serial manner and offloading the AIs in parallel. In the base cases, the game stops in the middle of the experiment, so we are not able to compute an average consistent with the method from the other experiments. However, since the client was removed from the game automatically, it must have exceeded the threshold of 200 ms repeatedly.

For quantity of data, we sum up the data transmitted over the network, both sent and received, over a period of 8 minutes that represents the maximum period for which we have data in all 4 cases. In this sum, we exclude the initialization phase, which does not belong to the processing loop. To estimate the analytical result, we assume an average of 10 iterations per second, and thus a total of 4800 of round-trips, which include sending to server data of 40 bytes each and receiving 120 bytes.

Table 5.2: Comparison between empirical and analytical results

Scenario	Average iteration time		Total data in 8 min	
	T [ms]		Q [bytes]	
	empirical	analytical	empirical	analytical
No network	>200	265.64	64,211	0
No offloading	>200	311.05	1,307,300	768,000
AI serial offloading	155.7	135.45	1,463,240	768,000
AI parallel offloading	154.41	119.78	1,473,267	768,000

It can be noted that our observation, that running 4 AI players is prohibitive for clients that do not offload, is consistent for both methods. However, this is also the cause why we were not able to assess empirically the exact performance for the base cases. On the other hand, serial and parallel offloading do not seem to vary significantly for AIs, although our analytical evaluation showed that parallel offloading should be able to bring an additional 9% speedup. Still, as expressed in Section 5.2, we find a significant improvement, of approximately 5%, in terms of in-game time.

In terms of data transfers, the analytical results do not match the empirical results, probably because in reality there are more information passed between client and server than the commands. In the no network case, we can see an overhead of 64kB of data that seems to be caused by other applications, since OpenTTD has no network enabled. We consider this background noise, having two orders of magnitude less than the real messages, and relatively constant, as it is probably caused by services that synchronize in the background. Even when substracting this value from the empirical values, they still remain significantly higher than the ones from the analytical evaluation. Therefore, it seems that OpenTTD also transmits other data and this data is larger when offloading AIs. However, data for serial and parallel AI offloading does not differ much, which is consistent with the analytical results.

We find that the operational analysis is a promising tool. Although the actual values are significantly different than the empirical ones, many of the ideas behind the offloading mechanisms are supported by both sets of results.

Chapter 6

Conclusion

This thesis focuses on defining an exploratory space for offloading concerns and using it in the analysis and empirical evaluation of various offloading mechanisms for mobile devices. To accomplish this, in Chapter 2 we structure the vast information from the research literature on offloading for mobiles and we propose a Taxonomy and an Exploratory Space for offloading mechanisms. Then, in Chapter 3 we propose a workload model for online social applications and we use it to characterize and model traces we collected for thousands of Facebook applications and several native mobile applications. In Chapter 4 we investigate in detail how various offloading mechanisms work on several types of applications and propose a formalism that can be used to conduct operational analysis of the offloading mechanisms in the Exploratory Space, applied to any application that functions by iterating over a processing loop. Finally, in Chapter 5 we validate our formalism by comparing analytical evaluation results with empirical evaluation results.

We believe that focusing on a specific type of application can bring advances in offloading for mobile devices, while still keeping a wide range of applicability. The idea of applying this research specifically to online social applications is *new*. From the point of view of the technology used to implement the applications, our approach covers both *web-based applications*, applications that send messages with the user actions to be performed on the server, and *tightly coupled simulations*, which do some heavy processing on the device and usually communicate only control messages. Through our workload model (Section 3.3) and our operational analysis (Section 4.3) we generalize both types of technologies and refer to any kind of loop-based application. We conduct an innovative design space exploration, based on the exploratory space we propose in Section 2.4.2, through empirical and analytical evaluation.

Our results on workload modeling have been received with great interest by the scientific community, as shown by the articles we published on this topic [79][80], and by the Best Paper Award we received at the 13th International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2013) in the Netherlands. The exploratory space is comprehensive and versatile, given the numerous contributions up to this date and the numerous opportunities for future research.

6.1 Research Contributions

The thesis includes multiple theoretical and practical contributions in the research area of offloading for mobile devices. The main contributions of the thesis are the characterization and modeling of workloads of online social applications and the design space exploration we perform through empirical and analytical evaluation of various offloading mechanisms.

In more detail, the contributions of this thesis are to:

- propose an Exploratory Space (Section 2.4.2) that shows novel offloading mechanisms, such as parallel and partial offloading in loop-based applications, as well as the opportunity of a design space exploration;
- collect traces of online social applications (Section 3.2) that contain information about workloads at macro-level (number of users) and at micro-level (user operations) for 16.000 Facebook games and several native mobile applications;
- propose a workload model (Section 3.3) with several components: Popularity Distribution and Evolution Pattern, for macro-level information, Packet-Level Traces and State Transition Diagram, for micro-level information;
- conduct characterization and modeling (Section 3.4) using the traces of online social applications on the workload model, showing that normal applications, with a single peak in user numbers, reach maturity early in their lifetimes, and that social networking features are increasingly influencing the user experience of online applications;
- show how our workload model can be used (Section 3.5), to predict traffic and evolution typology, as well as to generate synthetic traces;
- propose an Adaptive Query Algorithm for location-oriented applications, as a communication adaptation mechanism, and show what practical benefits it can have in a real application that monitors air quality using a sensor device (Section 4.1);
- propose a multi-layer architecture that enables mobiles connect to a home automation network using an USB dongle, as a communication offloading mechanism (Section 4.1);
- design and implement an Augmented Reality application, that overlays QR codes with multimedia in a loop-based process, and experiment with computation adaptation to improve its performance;
- modify a popular open-source game, that works by iterating over a processing loop, and use it to experiment with computation offloading;
- propose a formalism (Section 4.3) for operational analysis of the offloading mechanisms in the Exploratory Space, which can be used for any kind of application that functions by iterating over a processing loop;
- design and implement a testbed based on a real-world application and use the testbed to conduct empirical evaluation for some of the offloading mechanisms in the Exploratory Space (Section 5.2;

- generate synthetic workloads using the workload model and use them to conduct analytical evaluation for some of the offloading mechanisms in the Exploratory Space (Section 5.3);

6.2 Future Directions

Offloading for mobile devices is a generous topic. We identify here some of the directions in which further contributions can be made, especially related to workload modeling for online social application and to offloading mechanisms for loop-based application.

The workload model can be improved for better accuracy and for covering more applications. First, we propose studying the Evolution Pattern using time series analysis principles. According to the time series analysis theory, we study the evolution patterns under three components (the trend, the seasonal/cyclical component and the irregular component). The irregular component covers the bursts that some of the applications display during their evolution. Removing the irregular and seasonal components leaves a smoother trend component, which can be approximated with better results by either the linear model or by more complex distributions [45]. Second, we already have in mind new elements of the workload model that can take into consideration user actions. Third, we continue to collect data and we believe that using the workload model on even larger datasets will increase its accuracy.

We plan to propose an analytical model based on queuing theory [74] to better match the processing flow in offloading systems based on client-server architectures. The analytical model can easily and accurately provide more metrics of an offloading system, such as availability and reliability. We also plan to extend our focus on the computational infrastructure load with possible results in capacity planning and provisioning. Both directions can serve as basis for broader analytical evaluation of offloading systems and can be validated through empirical evaluation.

As mobile applications and mobile users are evolving, research efforts on offloading for mobile devices need to evolve as well. For example, until a few years ago multi-threading, although available at the programming level, did not have any impact on performance, because most of the CPUs were single core. In recent years mobile devices became as sophisticated as personal computers, with multi-core CPUs and some form of GPU. Therefore, current offloading research needs to take into consideration multi-threading and parallelism on the device when offloading. Also, it is foreseeable that, in the near future, even smaller and less capable computing devices, in the form wearable objects, will become increasingly popular, so mobile devices should be seen not as a source, but as a target for offloading.

List of Publications

Awards and Nominations

1. **Best Doctoral Symposium Paper** at CCGrid 2013, May 2013, The Netherlands, for the paper: 'Extending the capabilities of mobile devices for online social applications through cloud offloading'

2. Nomination for **Best Poster Award** at CCGrid 2013, May 2013, The Netherlands, for the poster following the paper mentioned above

3. **Research Grant** from the Romanian American Foundation to continue the Adaptive Query Algorithm presented in Section 4.1 during Nov 2013 - May 2014

Conference Proceedings Papers

1. A. Gherghina, **A. C. Olteanu**, and N. Ţăpuş. A marker-based augmented reality system for mobile devices. In *RoEduNet*, 2013. Available: http://bit.ly/XXPLYV

2. A. Făsui, **A. C. Olteanu**, and N. Ţăpuş. Fault tolerant surveillance system based on a network of mobile devices. In *RoEduNet*, 2013. Available: http://bit.ly/WCOr0K

3. V. Zamfirache, **A. C. Olteanu**, and N. Ţăpuş. Collaborative learning assistant for android. In *RoEduNet*, 2013. Available: http://bit.ly/11cQ9u3

4. D. O. Rizea, D. Ş. Tudose, **A. C. Olteanu**, and N. Ţăpuş. Adaptive query algorithm for location oriented applications. In *RoEduNet*, 2013. Available: http://bit.ly/WGXRbg

5. **A.C. Olteanu**, G. D. Oprina, N. Ţãpuş, and S. Zeisberg. Enabling mobile devices for home automation using zigbee. In *CSCS*, pages 189–195. IEEE, 2013

6. **A. C. Olteanu**, A. Iosup, and N. Ţãpuş. Towards a workload model for online social applications: Icpe 2013 work-in-progress paper. In *ICPE*, pages 319–322. ACM, 2013

7. **A. C. Olteanu**, N. Ţăpuş, and A. Iosup. Extending the capabilities of mobile devices for online social applications through cloud offloading. In *CCGRID*, pages 160–163, 2013

8. M.A. Popescu, O. Sluşanschi, and **A.C. Olteanu**. New bounds of a measure in information theory. In *Dezvoltare durabilã în condiţii de instabilitate economicã*, 2013
9. S. Chiricescu, **A.C. Olteanu**, and N. Ţăpuş. Interacţiunea cu dispozitive mobile pe baza detecţiei mişcãrilor feţei. In *RoCHI*, 2013. In publication. Available:
10. A. Gherghina, **A. C. Olteanu**, and N. Ţăpuş. Measuring performance in application offloading for mobile devices. In *ICSCS*, 2013. In publication. Available:
11. A. Făsui, **A. C. Olteanu**, and N. Ţăpuş. A survey regarding grid and distributed computing using mobile devices. In *ICSCS*, 2013. In publication. Available:
12. V. Zamfirache, A. Eftenoiu, P. Iosif, **A. C. Olteanu**, and N. Ţăpuş. Extending the moodle course management system for mobile devices. In *ICSCS*, 2013. In publication. Available:
13. **A. C. Olteanu**, D. S. Tudose, and N. Ţăpuş. Energy-efficient user interaction with an off-grid building. In *ICSCS*, 2013. In publication. Available:
14. R. Prejbeanu, **A. C. Olteanu**, and N. Ţăpuş. Real time collaboration in cloud for tune-up android. In *RoEduNet (Fall)*, 2013. In publication. Available:

Journal Articles

1. **A. C. Olteanu**, A. Iosup, N. Ţăpuş, and F. Kuipers. A workload evolution model for online social games. *Internet Computing*. submitted
2. **A. C. Olteanu** and N. Ţăpuş. Offloading for mobile devices: A survey. *UPB Scientific Bulletin*. submitted

Presentations at Scientific Events

1. **A. C. Olteanu**, D. S. Tudose, A. Voinescu, and N. Ţăpuş. Complete hardware and software solution for energy economy. In *WERC*, 2012
2. G. Oprina, **A.C. Olteanu**, D. S. Tudose, and N. Ţăpuş. Enabling mobile devices with medical diagnosis capabilities. In *WPA*, 2013
3. D. Rizea, D. S. Tudose, **A. C. Olteanu**, and N. Ţăpuş. Mobile application for pollution data. In *WPA*, 2013
4. **A. C. Olteanu**, R. I. Chioibasu, and N. Ţăpuş. Design of m-health solutions for psychology practitioners. In *WPA*, 2013‘

Technical and Scientific Reports

1. **A. C. Olteanu**, A. Iosup, and N. Ţăpuş. Towards a workload model for online

social applications: Extended report. Tech.Rep. PDS-2013-003, TU Delft, January 2013

2. **A. C. Olteanu**, A. Iosup, and N. Ţăpuş. Cloud offloading for mobile computing: A survey. Scientific report, UPB, 2013
3. **A. C. Olteanu**, A. Iosup, and N. Ţăpuş. Workload modeling for online social applications on mobile devices. Scientific report, UPB, 2013

Bibliography

[1] Mark Weiser, Rich Gold, and John Seely Brown. The origins of ubiquitous computing research at parc in the late 1980s. *IBM systems journal*, 38(4):693–696, 1999.

[2] CNN. Facebook reaches one billion users, 2012.

[3] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, pages 301–314. ACM, 2011.

[4] Tim Verbelen, Pieter Simoens, Filip De Turck, and Bart Dhoedt. Aiolos: Middleware for improving mobile application performance through cyber foraging. *Journal of Systems and Software*, 85(11):2629–2639, 2012.

[5] P. Ghosh, N. Roy, and S.K. Das. Mobility-aware efficient job scheduling in mobile grids. In *CCGrid*. IEEE, 2007.

[6] K.A. Hummel and G. Jelleschitz. A robust decentralized job scheduling approach for mobile peers in ad-hoc grids. In *CCGrid*. IEEE, 2007.

[7] D. Bruneo, M. Scarpa, A. Zaia, and A. Puliafito. Communication paradigms for mobile grid users. In *CCGrid*. IEEE, 2003.

[8] A.T.A. Gomes, A. Ziviani, L.S. Lima, and M. Endler. Dichotomy: A resource discovery and scheduling protocol for multihop ad hoc mobile grids. In *CCGrid*. IEEE, 2003.

[9] Huber Flores and Satish Nayarana Srirama. Adaptive code offloading and resource-intensive task delegation for mobile cloud applications.

[10] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *IEEE Perv. Comp.*, 2009.

[11] Rajesh Krishna Balan, Darren Gergle, Mahadev Satyanarayanan, and James Herbsleb. Simplifying cyber foraging for mobile devices. In *Proceedings of the 5th international conference on Mobile systems, applications and services*, MobiSys '07, pages 272–285. ACM, 2007.

[12] Mads Darø Kristensen and Niels Olof Bouvin. Scheduling and development support in the scavenger cyber foraging system. *Pervasive and Mobile Computing*, 6(6):677–692, 2010.

[13] Facebook. Facebook reaches one billion users, 2012.

[14] 10 ways cloud computing will change in 2013.

[15] S. Ou, K. Yang, and J. Zhang. An effective offloading middleware for pervasive services on mobile devices. *Pervasive and Mobile Computing*, 3(4):362–385, 2007.

[16] Georgios Portokalidis, Philip Homburg, Kostas Anagnostakis, and Herbert Bos. Paranoid android: versatile protection for smartphones. In *Proceedings of the 26th Annual Computer Security Applications Conference*, pages 347–356. ACM, 2010.

[17] Huber Flores, Satish Narayana Srirama, and Carlos Paniagua. A generic middleware framework for handling process intensive hybrid cloud services from mobiles. In *Proceedings of the 9th International Conference on Advances in Mobile Computing and Multimedia*, pages 87–94. ACM, 2011.

[18] Heungsik Eom, Pierre St Juste, Renato Figueiredo, Omesh Tickoo, Ramesh Illikkal, and Ravishankar Iyer. Snarf: a social networking-inspired accelerator remoting framework. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, MCC '12, pages 29–34. ACM, 2012.

[19] Yu-Ju Hong, K. Kumar, and Yung-Hsiang Lu. Energy efficient content-based image retrieval for mobile systems. In *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on*, pages 1673 –1676, may 2009.

[20] Tim Verbelen, Pieter Simoens, Filip De Turck, and Bart Dhoedt. Cloudlets: Bringing the cloud to the mobile user. In *Proceedings of the third ACM workshop on Mobile cloud computing and services*, pages 29–36. ACM, 2012.

[21] Ioana Giurgiu, Oriana Riva, Dejan Juric, Ivan Krivulev, and Gustavo Alonso. Calling the cloud: enabling mobile phones as interfaces to cloud applications. In *Middleware 2009*, pages 83–102. Springer, 2009.

[22] Gonzalo Huerta-Canepa and Dongman Lee. A virtual cloud computing provider for mobile devices. In *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing &#38; Services: Social Networks and Beyond*, MCS '10, pages 6:1–6:5, New York, NY, USA, 2010. ACM.

[23] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 49–62. ACM, 2010.

[24] Xiaohui Gu, Klara Nahrstedt, Alan Messer, Ira Greenberg, and Dejan Milojicic. Adaptive offloading for pervasive computing. *Pervasive Computing, IEEE*, 3(3):66–73, 2004.

[25] Lide Zhang, Birjodh Tiwana, Zhiyun Qian, Zhaoguang Wang, Robert P Dick, Zhuoqing Morley Mao, and Lei Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 105–114. ACM, 2010.

[26] Xinwen Zhang, Sangoh Jeong, Anugeetha Kunjithapatham, and Simon Gibbs. Towards an elastic application model for augmenting computing capabilities of mobile platforms. In *Mobile wireless middleware, operating systems, and applications*, pages 161–174. Springer, 2010.

[27] Radu-Corneliu Marin and Ciprian Dobre. Reaching for the clouds: contextually enhancing smartphones for energy efficiency. 2013.

[28] M. Ferber, T. Rauber, M.H.C. Torres, and T. Holvoet. Resource allocation for cloud-assisted mobile applications. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 400 –407, june 2012.

[29] Tingxin Yan, Vikas Kumar, and Deepak Ganesan. Crowdsearch: exploiting crowds for accurate real-time image search on mobile phones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 77–90. ACM, 2010.

[30] Karthik Kumar and Yung-Hsiang Lu. Cloud computing for mobile users: Can offloading computation save energy? *IEEE Computer*, 43(4):51–56, 2010.

[31] E. Lagerspetz and S. Tarkoma. Mobile search and the cloud: The benefits of offloading. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2011 IEEE International Conference on*, pages 117 –122, march 2011.

[32] Huber Raul Flores Macario and Satish Srirama. Adaptive code offloading for mobile cloud applications: exploiting fuzzy sets and evidence-based learning. In *Proceeding of the fourth ACM workshop on Mobile cloud computing and services*, pages 9–16. ACM, 2013.

[33] Mohammed Anowarul Hassan and Songqing Chen. Mobile mapreduce: Minimizing response time of computing intensive mobile applications. In *MobiCASE*, pages 41–59, 2011.

[34] Andreas Klein, Christian Mannweiler, Joerg Schneider, and Hans D Schotten. Access schemes for mobile cloud computing. In *Mobile Data Management (MDM), 2010 Eleventh International Conference on*, pages 387–392. IEEE, 2010.

[35] Dinh Thai Hoang, Dusit Niyato, and Ping Wang. Optimal admission control policy for mobile cloud computing hotspot with cloudlet. In *Wireless Communications and Networking Conference (WCNC), 2012 IEEE*, pages 3145–3149. IEEE, 2012.

[36] Roelof Kemp, Nicholas Palmer, Thilo Kielmann, and Henri Bal. Cuckoo: A computation offloading framework for smartphones. In *Mobile Computing, Applications, and Services*, volume 76 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 59–79. Springer Berlin Heidelberg, 2012. 10.1007/978-3-642-29336-8_4.

[37] Google, Inc. *Android Cloud to Device Messaging Framework*, 2012.

[38] R. Kemp, N. Palmer, T. Kielmann, and H. Bal. Energy efficient information monitoring applications on smartphones through communication offloading. *Mobicase*, 2012.

[39] Mahadev Satyanarayanan. Pervasive computing: Vision and challenges. *Personal Communications, IEEE*, 8(4):10–17, 2001.

[40] Mahadev Satyanarayanan. Mobile computing: the next decade. *ACM SIGMOBILE Mobile Computing and Communications Review*, 15(2):2–10, 2011.

[41] Shaoxuan Wang and Sujit Dey. Rendering adaptation to address communication and computation constraints in cloud mobile gaming. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pages 1–6. IEEE, 2010.

[42] M.F. Arlitt and C.L. Williamson. Web server workload characterization: The search for invariants. In *ACM SIGMETRICS*, volume 24, pages 126–137. ACM, 1996.

[43] L. Cherkasova, Y. Fu, W. Tang, and A. Vahdat. Measuring and characterizing end-to-end internet service performance. *ACM TOIT*, 3(4):347–391, 2003.

[44] A.K. Iyengar, M.S. Squillante, and L. Zhang. Analysis and characterization of large-scale web server access patterns and performance. *WWW*, 2(1):85–100, 1999.

[45] B. Zhang, A. Iosup, J. Pouwelse, and D. Epema. Identifying, analyzing, and modeling flashcrowds in bittorrent. In *P2P*, pages 240–249. IEEE, 2011.

[46] M. Suznjevic and M. Matijasevic. Player behavior and traffic characterization for mmorpgs: a survey. *Multimedia Systems*, pages 1–22, 2012.

[47] A.K. Iyengar, E.A. MacNair, M.S. Squillante, and L. Zhang. A general methodology for characterizing access patterns and analyzing web server performance. In *MASCOTS*, pages 167–174. IEEE, 1998.

[48] D. Kondo, B. Javadi, A. Iosup, and D. Epema. The failure trace archive: Enabling comparative analysis of failures in diverse distributed systems. In *CCGrid*, pages 398–407. IEEE, 2010.

[49] N. Yigitbasi, M. Gallet, D. Kondo, A. Iosup, and D. Epema. Analysis and modeling of time-correlated failures in large-scale distributed systems. In *GRID*, pages 65–72. IEEE, 2010.

[50] Y. Guo and A. Iosup. The game trace archive. In *NetGames*, pages 1–6. IEEE, 2012.

[51] Y. Guo, S. Shen, O. Visser, and A. Iosup. An analysis of online match-based games. In *HAVE*, pages 134–139. IEEE, 2012.

[52] A. Nazir, S. Raza, and C.N. Chuah. Unveiling facebook: a measurement study of social network based applications. In *IMC*, 2008.

[53] A. Nazir, S. Raza, D. Gupta, C.N. Chuah, and B. Krishnamurthy. Network level footprints of facebook applications. In *IMC*. ACM, 2009.

[54] B. Kirman, S. Lawson, and C. Linehan. Gaming on and off the social graph: The social structure of facebook games. In *CSE*, volume 4, pages 627–632. IEEE, 2009.

[55] Onlive has over 1.5 million active users.

[56] R. Prejbeanu, **A. C. Olteanu**, and N. Ţăpuş. Real time collaboration in cloud for tune-up android. In *RoEduNet (Fall)*, 2013. In publication. Available: .

[57] StatSoft. How to identify patterns in time series data: Time series analysis.

[58] Danielle Bilodeau. Seasonal adjustment: why, when, how? Institut de la statistique du Québec.

[59] Aaron Carroll and Gernot Heiser. An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, pages 21–21, 2010.

[60] D. O. Rizea, D. Ş. Tudose, **A. C. Olteanu**, and N. Ţăpuş. Adaptive query algorithm for location oriented applications. In *RoEduNet*, 2013. Available: http://bit.ly/WGXRbg.

[61] D. Rizea, D. S. Tudose, **A. C. Olteanu**, and N. Ţăpuş. Mobile application for pollution data. In *WPA*, 2013.

[62] Gian Paolo Perrucci, Frank HP Fitzek, and Jörg Widmer. Survey on energy consumption entities on the smartphone platform. In *Vehicular Technology Conference (VTC Spring), 2011 IEEE 73rd*, pages 1–6. IEEE, 2011.

[63] Choong-Bum Park, Byung-Sung Park, Huy-Jung Uhm, Hoon Choi, and Hyong-Shik Kim. Ieee 802.15. 4 based service configuration mechanism for smartphone. *Consumer Electronics, IEEE Transactions on*, 56(3):2004–2010, 2010.

[64] Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 280–293. ACM, 2009.

[65] Ye-Sheng Kuo. Hijacking power and bandwidth from the mobile phone's audio interface. In *Proceedings of the First ACM Symposium on Computing for Development.* ACM, 2010.

[66] Damien Phelan Stolarz, George Dean IV, and Zack Gainsforth. Electronic device input/output system and method, March 8 2010. US Patent App. 12/660,995.

[67] A. Gherghina, **A. C. Olteanu**, and N. Ţăpuş. A marker-based augmented reality system for mobile devices. In *RoEduNet*, 2013. Available: http://bit.ly/XXPLYV.

[68] Trevor Pering, Yuvraj Agarwal, Rajesh Gupta, and Roy Want. Coolspots: reducing the power consumption of wireless mobile devices with multiple radio interfaces. In *Proceedings of the 4th international conference on Mobile systems, applications and services*, pages 220–232. ACM, 2006.

[69] Wei Yu, George Ginis, and John M Cioffi. An adaptive multiuser power control algorithm for vdsl. In *Global Telecommunications Conference, 2001. GLOBECOM'01. IEEE*, volume 1, pages 394–398. IEEE, 2001.

[70] Anant Agarwal and Mathews Cherian. *Adaptive backoff synchronization techniques*, volume 17. ACM, 1989.

[71] Tijs Van Dam and Koen Langendoen. An adaptive energy-efficient mac protocol for wireless sensor networks. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 171–180. ACM, 2003.

[72] Yong-Soon Mun. Home automation system having user controlled definition function, November 26 1996. US Patent 5,579,221.

[73] Thad Starner, Jake Auxier, Daniel Ashbrook, and Maribeth Gandy. The gesture pendant: A self-illuminating, wearable, infrared computer vision system for home

automation control and medical monitoring. In *Wearable Computers, The Fourth International Symposium on*, pages 87–94. IEEE, 2000.

[74] Leonard Kleinrock. Queueing systems. volume 1: Theory. 1975.

[75] Daniel A Menasce, Virgilio AF Almeida, Lawrence Wilson Dowdy, and Larry Dowdy. *Performance by design: computer capacity planning by example.* Prentice Hall Professional, 2004.

[76] Raj Jain. *The art of computer systems performance analysis*, volume 182. John Wiley & Sons Chichester, 1991.

[77] Peter JB King. *Computer and communication systems performance modelling.* Prentice Hall International (UK) Ltd., 1990.

[78] A. Gherghina, **A. C. Olteanu**, and N. Ţăpuş. Measuring performance in application offloading for mobile devices. In *ICSCS*, 2013. In publication. Available: .

[79] **A. C. Olteanu**, A. Iosup, and N. Ţãpuş. Towards a workload model for online social applications: Icpe 2013 work-in-progress paper. In *ICPE*, pages 319–322. ACM, 2013.

[80] **A. C. Olteanu**, A. Iosup, N. Ţăpuş, and F. Kuipers. A workload evolution model for online social games. *Internet Computing.* submitted.

[81] A. Făsui, **A. C. Olteanu**, and N. Ţăpuş. Fault tolerant surveillance system based on a network of mobile devices. In *RoEduNet*, 2013. Available: http://bit.ly/WCOr0K.

[82] V. Zamfirache, **A. C. Olteanu**, and N. Ţăpuş. Collaborative learning assistant for android. In *RoEduNet*, 2013. Available: http://bit.ly/11cQ9u3.

[83] **A.C. Olteanu**, G. D. Oprina, N. Ţãpuş, and S. Zeisberg. Enabling mobile devices for home automation using zigbee. In *CSCS*, pages 189–195. IEEE, 2013.

[84] **A. C. Olteanu**, N. Ţăpuş, and A. Iosup. Extending the capabilities of mobile devices for online social applications through cloud offloading. In *CCGRID*, pages 160–163, 2013.

[85] M.A. Popescu, O. Sluşanschi, and **A.C. Olteanu**. New bounds of a measure in information theory. In *Dezvoltare durabilã în condiţii de instabilitate economicã*, 2013.

[86] S. Chiricescu, **A.C. Olteanu**, and N. Ţăpuş. Interacţiunea cu dispozitive mobile pe baza detecţiei mişcãrilor feţei. In *RoCHI*, 2013. In publication. Available: .

[87] A. Făsui, **A. C. Olteanu**, and N. Ţăpuş. A survey regarding grid and distributed computing using mobile devices. In *ICSCS*, 2013. In publication. Available: .

[88] V. Zamfirache, A. Eftenoiu, P. Iosif, **A. C. Olteanu**, and N. Ţăpuş. Extending the moodle course management system for mobile devices. In *ICSCS*, 2013. In publication. Available: .

[89] **A. C. Olteanu**, D. S. Tudose, and N. Ţăpuş. Energy-efficient user interaction with an off-grid building. In *ICSCS*, 2013. In publication. Available: .

[90] **A. C. Olteanu** and N. Ţăpuş. Offloading for mobile devices: A survey. *UPB Scientific Bulletin.* submitted.

[91] **A. C. Olteanu**, D. S. Tudose, A. Voinescu, and N. Ţăpuş. Complete hardware and software solution for energy economy. In *WERC*, 2012.

[92] G. Oprina, **A.C. Olteanu**, D. S. Tudose, and N. Ţăpuş. Enabling mobile devices with medical diagnosis capabilities. In *WPA*, 2013.

[93] **A. C. Olteanu**, R. I. Chioibasu, and N. Ţăpuş. Design of m-health solutions for psychology practitioners. In *WPA*, 2013.

[94] **A. C. Olteanu**, A. Iosup, and N. Ţăpuş. Towards a workload model for online social applications: Extended report. Tech.Rep. PDS-2013-003, TU Delft, January 2013.

[95] **A. C. Olteanu**, A. Iosup, and N. Ţăpuş. Cloud offloading for mobile computing: A survey. Scientific report, UPB, 2013.

[96] **A. C. Olteanu**, A. Iosup, and N. Ţăpuş. Workload modeling for online social applications on mobile devices. Scientific report, UPB, 2013.