Graduate School ETD Form 9
(Revised 12/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Pelin Angin

Entitled
Autonomous Agents-Based Mobile-Cloud Computing

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

Bharat Bhargava
Chair

Vernon J. Rego

Luo Si

Aditya P. Mathur

Buster Dunsmore

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): Bharat Bhargava

Approved by: Sunil Prabhakar / William J. Gorman 11/26/2013
Head of the Graduate Program Date

AUTONOMOUS AGENTS-BASED MOBILE-CLOUD COMPUTING

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Pelin Angin

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2013

Purdue University

West Lafayette, Indiana

To my family

ACKNOWLEDGMENTS

This dissertation would not have been possible without the continuous support and guidance of my advisor, Professor Bharat Bhargava. Getting to work under Professor Bhargava's supervision has been one of the greatest chances in my life, as he not only taught me a lot about academia and prepared me for the professional life, but also provided great moral support, which always motivated me to do better. His enthusiasm for sharing ideas and helping others, as well as his unwavering belief in my potential very much influenced the decisions I made, and helped me to never give up. Thanks to his diverse research interests and his being on the lookout for the most significant research problems, I had the chance to be part of various research projects and deepen my knowledge in various fields of computer science. I am also grateful to Mrs. Shail Bhargava for all her support and hospitality throughout my graduate study.

I would also like to express my deepest gratitude to my advisory committee members, Professor Aditya Mathur, Professor Luo Si and Professor Vernon Rego. Their feedback and support helped me steer my research on the right path and stay focused. I cannot thank Professor Buster Dunsmore enough for not only being in my final examination committee, but also for all the support he gave me ever since I became a teaching assistant for his software engineering course. I was very lucky to work under his supervision and l really learnt a lot from him about excellence in both teaching and human values.

I am grateful to all faculty members at the Purdue Department of Computer Science for making our department a top-notch one in the nation, and especially Profs. Sonia Fahmy, Sunil Prabhakar, Cristina Nita-Rotaru, Patrick Eugster and Greg Frederickson, who made the courses I took with them a great experience. I owe special thanks to Professor Jennifer Neville for teaching me how to conduct research

and always supporting me in the decisions I made in the early years of my graduate study. I am also thankful to Professor Sumi Helal from the University of Florida and Professor Leszek Lilien from Western Michigan University for their collaboration and meticulous feedback. I am also indebted to the staff at the Computer Science Department, including Dr. William J. Gorman, Renate Mallus-Medot, Sandra Freeman, Amy Ingram, Pam Graf, Dr. Tim Korb and Jean Jackson for making our department such a warm and welcoming environment.

I am thankful to my labmates, Rohit Ranchal, Ruchith Fernando, Mehdi Azarmi and Mark (Zhongjun) Jin for their great support and collaboration on many projects. I would also like to acknowledge the invaluable contributions of Dr. Lotfi ben Othmane, Dr. Nwokedi Idika, Norman Ahmed and Ranjitkumar Sivakumar to the research activities of our group. I am also grateful to Dr. Cary Supalo and Mick Isaacson for all the time they spared to discuss my research despite their busy schedule.

Life at Purdue would not have been half as good without some very special friends. I thank Vasil Stefanov Denchev, Rongjing Xiang, Chenyun Dai, Sevim Dalva and Nursel Cakir Yavuzarslan for being my family away from home. I am also thankful to Pinar Yanardag Delul, Ilke Demir, Suleyman Cetintas, Hasan Onan Demirel, Ozgur Keles, Kerem Camsari, Debbie Perouli, Camille Gaspard, Chamikara Jayalath, Guneshi Thanuja Wickramaarachchi, Carlos Vanegas, Umang Sharan, Serkan Uzunbaz, Ahmet Erhan Nergiz, Ahmet Bugdayci and many other friends who brightened my days with their friendship. I would also like to acknowledge the importance of the Turkish Student Association in my social life at Purdue, and especially the constant support of Nejla Yazici, Huseyin Aydin and Professor Huseyin Gulen. I am also grateful to all employees of the Greyhouse Cafe for helping me through the sleepless nights with their wonderful coffee and always smiling faces.

I owe sincere thanks to all faculty members at the Computer Engineering Department at Bilkent University, where I got my undergraduate education. It would not have been possible for me to make the right choices about my graduate study and get admitted to a top-notch university without the support and guidance of notable

Profs. Ozgur Ulusoy, Ibrahim Korpeoglu, Pinar Duygulu Sahin and Tugrul Dayar. I am also grateful to Profs. Fazli Can, David Davenport, Ali Aydin Selcuk and Sengor Altingovde for the depth of knowledge they gave me during my study at Bilkent. I cannot thank the T.E.D. Ankara College Foundation enough for the excellent primary and secondary education I got there, where I developed a passion for math and science and had the chance to learn English from native speakers.

Lastly, I would like to thank my family for always being there for me and the unconditional love they have given me. My mother deserves special thanks for being such a loving parent and all the sleepless nights she spent during my whole graduate study, and my father deserves the same for his patience and support. I am grateful to my sister for being the shoulder I cried on at hard times and being my best friend since she was born. I am thankful to my uncle and my grandparents for being such wonderful people, serving as role models for me throughout my life.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

ABBREVIATIONS

AAMCC	Autonomous Agents-based Mobile-Cloud Computing
AE	Authenticated Encryption
AMI	Amazon Machine Image
AOP	Aspect-Oriented Programming
EC2	Elastic Compute Cloud
IaaS	Infrastructure as a Service
JADE	Java Agent Development Environment
MAC	Message Authentication Code
MCC	Mobile-Cloud Computing
PaaS	Platform as a Service
QoS	Quality of Service
TTP	Trusted Third Party
UML	Unified Modeling Language
VMI	Virtual Machine Instance

ABSTRACT

Angin, Pelin Ph.D., Purdue University, December 2013. Autonomous Agents-based Mobile-Cloud Computing. Major Professor: Bharat K. Bhargava.

The proliferation of cloud computing resources in recent years offers a way for mobile devices with limited resources to achieve computationally intensive tasks in real-time. The mobile-cloud computing paradigm, which involves collaboration of mobile and cloud resources in such tasks, is expected to become increasingly popular in mobile application development. While mobile-cloud computing is promising to overcome the computational limitations of mobile devices, the lack of frameworks compatible with standard technologies makes it harder to adopt dynamic mobile-cloud computing at large. In this dissertation, we present a dynamic code offloading framework for mobile-cloud computing, based on autonomous agents. Our approach does not impose any requirements on the cloud platform other than providing isolated execution containers, and it alleviates the management burden of offloaded code by the mobile platform using autonomous agent-based application partitions. We also investigate the effects of different runtime environment conditions on the performance of mobile-cloud computing, and present a simple and low-overhead dynamic makespan estimation model for computation offloaded to the cloud that can be integrated into mobile agents to enhance them with self-performance evaluation capability.

Offloading mobile computation to the cloud entails security risks associated with handing sensitive data and code over to an untrusted platform. Security frameworks for mobile-cloud computing are not very numerous and most of them focus only on privacy, and ignore the very important aspect of integrity. Perfect security is hard to achieve in real-time mobile-cloud computing due to the extra computational overhead introduced by complex security mechanisms. In this dissertation, we propose a dynamic tamper-resistance approach for protecting mobile computation offloaded to the cloud, by augmenting mobile agents with self-protection capability. The tamper-resistance framework achieves very low execution time overhead and is capable of detecting both load-time and runtime modifications to agent code.

Lastly, we propose novel applications of mobile-cloud computing for helping context-aware navigation by visually-impaired people. Specifically, we present the results of a feasibility study for using real-time mobile-cloud computing for the task of guiding blind users at pedestrian crossings with no accessible pedestrian signal.

1. INTRODUCTION

1.1 Motivation

Mobile computing devices have become increasingly popular during the past decade, replacing desktops and mainframes for daily computing needs. Many of these devices have limited processing power, storage and battery compared to their wall-socket-powered, tethered counterparts, which limits their capabilities for real-time, computationally-intensive applications such as image processing. Computation offloading to more powerful servers is the solution to provide these devices with the resources they need to achieve complex tasks. Cloud computing, emerging as a new computing paradigm in the recent years, offers computing resources to users on demand, obviating the need to actually own those resources. With increasing popularity and availability, cloud computing has the potential to fill the gap between the resource needs of mobile devices and availability of those resources, through the concept of *mobile-cloud computing* (MCC), which partitions mobile applications between mobile and cloud platforms for execution.

Most mobile applications today involve an inflexible split of computation between the mobile and cloud platforms, following the client-server paradigm with hardcoded interactions with the server. This inflexibility prevents applications from adapting to conditions such as high network latency, which could result in poor performance when cloud resources are preferred over computation on the device. For applications requiring large data transfers to the remote server, performing the computation on the device can have better performance than relying on processing by the remote server when the device's Internet connection is poor.

Achieving high performance with mobile-cloud computing is contingent upon an optimal partitioning of the mobile application components between the mobile and cloud platforms based on runtime conditions, as well as the dynamic monitoring of the performance of application components during their execution. Recent work on this problem has resulted in frameworks with various partitioning and optimization techniques. Most of these frameworks impose strict requirements on the cloud side, such as a full clone of the application code/virtual machine or special application management software, hindering wide applicability in public clouds.

One reason to avoid flexibility in computation partitioning is the security risks associated with using the cloud: Application components involving sensitive data would need different security measures for migration to the cloud than those executing on the mobile device. The lack of control on resources and multi-tenancy of different users' applications on the same physical machine make cloud platforms vulnerable to attacks. Possible security breaches in the cloud include leakage/modification of private data and modification of program code among others. Because mobile-cloud computing is still a young field of research, there are not many security mechanisms for code and data offloaded to the cloud from a mobile device. Security mechanisms relying heavily on the participation of the mobile platform are undesirable due to the extra communication and processing costs involved for the mobile platform. Existing strong security mechanisms for mobile code can also not be applied directly to the problem of MCC security, as they are cryptography-intensive (i.e. computationally-intensive), which could cause an MCC framework to fall short of satisfying the real-time requirements of a computation task.

Both the performance and security problems associated with MCC call for a generalized computation offloading framework that requires minimal involvement of the mobile platform for both performance and security monitoring of offloaded computation, where all dynamic decision-making processes are achieved with lightweight components.

1.2 Thesis Statement

In this dissertation, we demonstrate that autonomous (mobile) agents are effective tools for dynamic computation offloading in MCC. We specifically focus on the design of a generalized computation offloading framework, where offloaded code is transported between platforms using mobile agents that are capable of protecting their own integrity and evaluating their own runtime performance.

The thesis statement of this dissertation is:

Autonomous agents, when augmented with self-protection and self-performance evaluation capability, are effective tools for high-performance, secure mobile-cloud computing.

1.3 Dissertation Contributions

The main contributions of this dissertation are the following:

- Design, development and performance evaluation of an efficient computation offloading framework for mobile-cloud computing based on autonomous agents
- Design, development and performance evaluation of a low-overhead tamper-resistance approach for mobile-cloud computing
- Design, development and performance evaluation of a context-aware computation migration and execution approach for mobile-cloud computing
- A novel application of mobile-cloud computing to assistive technologies for the visually-impaired

1.4 Dissertation Organization

The rest of this dissertation is organized as follows:

Chapter 2. Related Work In this chapter, we provide a summary of related work in the field of mobile-cloud computing, with particular focus on (a) existing computation offloading techniques and their use in mobile-cloud computing, and (b) mechanisms to protect code offloaded to a foreign (cloud) platform for execution. We state the major shortcomings of the existing mobile-cloud computation offloading frameworks and identify essential features for a framework accounting for those shortcomings.

Chapter 3. Mobile-Cloud Computing (MCC) In this chapter, we provide a brief overview of mobile-cloud computing, needed for a better understanding of the concepts used in the subsequent chapters. We start with a brief description of cloud computing, including common service models and deployment models. We continue with a definition of mobile-cloud computing and discuss its advantages, applications and general architecture. Next, we discuss the major challenges dynamic mobile-cloud computing faces, which provides the motivation for the design of a computation offloading framework overcoming those challenges.

Chapter 4. An Autonomous Agents-based Computation Offloading Framework for MCC In this chapter, we introduce a novel computation offloading framework for MCC based on autonomous (mobile) agents. We start the chapter with a brief overview of mobile agents, the advantages of using mobile agents for MCC and the details of the workings of a popular mobile agent framework. Next, we present the details of our proposed computation offloading framework for MCC, named AAMCC. We conclude the chapter with the results of experiments using AAMCC for three real-world mobile applications and a comparison of the results with those of monolithic execution of the same applications on a mobile device.

Chapter 5. Tamper-Resistant Execution in the Cloud In this chapter, we introduce a tamper-resistance technique for code offloaded to the cloud using our proposed computation offloading framework. We start with a discussion of tamper-resistance and security issues with mobile agents. Next, we discuss two main ideas for mobile code and data protection–active bundles and software guards–inspiring the approach for tamper-resistance that we take in this work. Then, we present the details of the elements used in our approach and state how they are used together to form a security framework for AAMCC. We conclude the chapter with a performance evaluation of the proposed tamper-resistance approach in terms of its security capabilities and runtime overhead.

Chapter 6. Context-Aware Mobile-Cloud Computing In this chapter, we discuss the need for dynamic execution profiling for high-performance MCC and introduce our proposed models for network conditions-based computation offloading and runtime execution profiling on a cloud platform. We demonstrate through experiments with the mobile applications introduced in the earlier chapters that the proposed context-aware computation models are effective for achieving high-performance under varying network and cloud conditions.

Chapter 7. Application: Mobile-Cloud Navigation Guide for the Blind In this chapter, we discuss the use of MCC in assistive technologies, specifically in context-aware navigation by visually-impaired people. We start the chapter with a discussion of the need for assistive technologies by the visually-impaired, and present the details of a mobile-cloud computing framework that we propose for context-aware navigation. We introduce a mobile-cloud pedestrian crossing guide application for the blind and discuss its feasibility of use for real-time guidance. We conclude the chapter with a discussion of other mobile-cloud applications that could help visually-impaired people achieve better context-awareness in foreign environments.

Chapter 8. Summary This chapter concludes the dissertation with a summary of the main contributions and discussion of possible future extensions to the work.

2. RELATED WORK

2.1 Computation Offloading

Significant research efforts have been put into computation offloading since the early 1990s. Until 2000, the focus was on making offloading feasible, due to the limitations in wireless networks. Improvements in virtualization technology, availability of high network bandwidths and cloud computing infrastructures increased the feasibility of real-time computation offloading.

Initial efforts on mobile computation offloading focused mostly on the problem of developing robust frameworks for heterogeneous pervasive computation environments such as mobile ad-hoc networks (MANETs). Chu et al. [1] propose Roam, which is a seamless application framework for migrating a running application among heterogeneous devices. The Roam system is based on partitioning of an application and it selects the most appropriate adaptation strategy at the component level of the target platform. Gu et al. [2] propose an offloading inference engine using the *fuzzy control* model, to solve the problem of timely triggering of adaptive offloading and intelligent selection of an application partitioning policy. Gouveris et al. [3] propose a middleware-based programmable infrastructure that allows nodes of a MANET to download and activate required protocol and service software dynamically, which allows for alignment of the nodes' capabilities so that common services and protocols can be used among heterogeneous network nodes.

Chen et al. [4] present an approach for choosing which components of Java programs to offload. Their approach divides a Java program into methods and uses input parameters such as the size of methods to compute the cost of executing each method. An optimal partitioning decision is then made by comparing the local execution cost of each method with the remote execution cost, based on the network conditions. Wang and Li [5] present a computation offloading scheme on mobile devices based on a polynomial time algorithm, which partitions a program into distributed subprograms, where all physical memory references are mapped into the references of abstract memory locations. Hunt and Scott [6] present a distributed partitioning system called Coign, which automatically transforms a program into distributed applications without access to their source code. Coign constructs a graph model of the application's intercomponent communication through network profiling to find the best partitioning and applies the lift-to-front minimum-cut graph-cutting algorithm [7] to choose the partitions with the minimum communication cost.

Ou et al. [8] analyze the performance of offloading systems in wireless environments, considering the cases where the application is executed locally, offloaded with no failures, and offloaded in the presence of failure and recoveries. Their approach only re-offloads failed subtasks, which provides shortened execution time.

Tang and Cao [9] explain appropriate solutions for offloading in different environments, taking into account three common environmental changes. Upon connection status changes, the server in their model periodically checks the connection status with the client and maintains the execution status about its running tasks. When the client is reconnected, the server sends the execution results. If the client never reconnects, the server waits for a specific time interval, and deletes the tasks.

Although efforts in computation offloading have a long history, research in mobile-cloud computing is still at its infancy. Early work in dynamic mobile-cloud computing models includes CloneCloud [10] and MAUI [11], both of which partition applications using a framework that combines static program analysis with dynamic program profiling, and optimizes execution time and energy consumption on the mobile device using an optimization solver. The disadvantage of these approaches is that they require a copy of the whole application code/virtual machine at the remote execution site, which is both a quite strict requirement for public cloud machines and makes the application code vulnerable to analysis by malicious parties on the same platform. Yang et al. [12] propose a partitioning and execution framework specifically for mobile data stream applications. Huang et al. [13] propose a low-complexity offloading algorithm to minimize energy consumption on a mobile device, however they do not provide details of the system architecture. Park et al. [14] propose an offloading framework limited to JavaScript based applications. The recently proposed ThinkAir [15] framework provides better scalability and parallelism features than its predecessors, however it still requires the existence of the complete application code on the cloud server, exhibiting the disadvantages of CloneCloud and MAUI.

Xian et al. [16] propose a computation offloading method that does not require the estimation of makespan for each application component. They use online statistics of the application makespan to compute an optimal timeout, and the computation is offloaded to a remote server only if it is not completed on the device by the timeout. They show through experiments that this approach not only addresses the problem of inaccurate makespan estimation but also saves more energy than existing approaches.

Li et al. [17] propose a middleware framework based on mobile agents for deploying mobile computation in the cloud. The problem they address is enabling of mobile services in the cloud. The purpose of the framework they propose is to provide a simple model for cloud services running and migrating in the cloud smoothly. The general idea behind their proposal is to establish an environment supporting global invocation and addressing of mobile services and provide mechanisms to manage the services for transparent utilization, which is quite different from the problem we address in this work.

2.2 MCC Security

The two main approaches to provide MCC security are (a) ensuring the security of the cloud platform on which the mobile code will execute for all users of that platform and (b) ensuring the security of the mobile code and data sent to the cloud platform for execution, without relying on the trustworthiness of the cloud platform. Below we provide a review of related work for each of these approaches.

2.2.1 Virtualization Security

Lombardi and Di Pietro [18] propose a secure virtualization architecture, which monitors the integrity of guest and infrastructure components while remaining fully transparent to virtual machines and to cloud users. The proposed architecture can locally react to security breaches as well as notify a further security management layer of such events and has low overhead.

Szefer et al. [19] propose a virtualization architecture, which eliminates the hypervisor attack surface by enabling the guest virtual machines (VMs) to run natively on the underlying hardware while maintaining the ability to run multiple VMs concurrently. Their proposed system is based on four key ideas: (i) pre-allocation of processor cores and memory resources, (ii) use of virtualized I/O devices, (iii) minor modifications to the guest OS to perform all system discovery during bootup, and (iv) avoiding indirection by bringing the guest virtual machine in more direct contact with the underlying hardware. Hence, in their proposed approach, no hypervisor is needed to allocate resources dynamically, emulate I/O devices, support system discovery after bootup, or map interrupts and other identifiers.

Zhang et al. [20] propose a transparent approach to protect the privacy and integrity of customers' virtual machines on commodity virtualized infrastructures, even in the case of facing a complete compromise of the virtual machine monitor (VMM) and the management VM. The key of their approach is the separation of the resource management from security protection in the virtualization layer. A tiny security monitor is introduced underneath the commodity VMM using nested virtualization and provides protection to the hosted VMs. As a result, their approach allows virtualization software to handle complex tasks of managing leased VMs for the cloud, without compromising the security of users' data inside the VMs.

Li et al. [21] address the problem of providing a secure execution environment on a virtualized computing platform under the assumption of an untrusted management OS. They propose a virtualization architecture that provides a secure runtime environment, network interface, and secondary storage for a guest VM. The proposed architecture significantly reduces the trusted computing base of security-critical guest VMs, leading to improved security in an untrusted management environment. The proposed architecture relies on the forcing of all memory access to go through the hypervisor and making sure that the host operating system only sees an encrypted view of the guest operating systems.

Although solutions for secure virtualization are promising to make cloud platforms reliable components for MCC, their implementation is (at least currently) limited to a small set of providers. Without guarantees for the existence of such security services, the security of code offloaded to the cloud cannot be ensured. These approaches also do not provide mechanisms for the protection of code running on the same virtual machine as an attacker.

2.2.2 Mobile Code and Data Security

One approach taken by previous research [22, 23] in mobile code security and secret program execution is homomorphic encryption, which operates with encrypted functions obtained by homomorphic transformations. However this approach was only shown to be usable for polynomial and rational functions, which prevents it from providing a general solution to the problem of secret execution in the cloud.

Another popular approach for preventing malicious analysis of mobile code is obfuscation. A code obfuscator is a tool which repeatedly applies semantics-preserving code transformations to a program [24]. The obfuscator tries to make a program as incomprehensible as possible in order to protect it from being reverse-engineered or to protect a secret stored in the program from being discovered. Typically, code obfuscating transformations use the following operations: fold/flatten (turn a d-dimensional construct into d+1 or d-1 dimensional ones), split/merge (turn a compound construct into two constructs or merge two constructs into one), box/unbox (add or remove a layer of abstraction), ref/deref (add or remove a level of indirection), reorder (swap two adjacent constructs), and rename (assign a new name to a labeled construct). While obfuscation makes a program more difficult to analyze, it comes at a cost: The obfuscated programs usually have greater complexity than their original, optimized counterparts, which causes worse performance. Therefore, the best practice for providing code confidentiality would be to use obfuscation sparingly, only in parts of a program, which include content that must be protected against leakage. As the obfuscation process itself could be costly too, it should be applied once for a mobile application before or upon installation instead of before each execution.

A recent proposal for securing cloud MANET applications [25] adopts cloud computing technology to create a virtualized environment for MANET operations in multiple service provisioning domains according to the criticality of the MANET services and corresponding security requirements.

Zhang et al. [26] propose an authentication and secure communication framework for elastic applications, which consist of weblets (program partitions) running on mobile device and cloud nodes concurrently. Their proposed method leverages elasticity managers on the cloud and the mobile device to establish a shared secret between weblets, to provide authentication between the weblets. They also propose approaches to authorize weblets running on the cloud to access sensitive data such as those provided by other web services. While this is one of the few proposals considering secure mobile-cloud computing, the authors do not mention any implementation details or performance results.

Itani et al. [27] propose a scheme to verify the integrity of documents stored on a cloud server. Their model is based on offloading most of the integrity verification tasks to the cloud and a trusted third party (TTP) to minimize the processing overhead and energy consumption on the mobile client. The main limitation of the proposed framework is that it is dependent on a TTP.

Jia et al. [28] propose a data service, which uses proxy re-encryption and identity-based encryption to outsource data and security management to the cloud without disclosing any user information. The main limitation of this work is that it trusts the cloud to perform the security management and re-encryption on behalf of the mobile user, while this trustworthiness assumption may not always hold.

Hsueh et al. [29] propose a security scheme for the storage of mobile data in the cloud that ensures the security, integrity, and authentication of the stored data. The proposed scheme is based on the encryption of the data by the mobile device using traditional asymmetric encryption techniques. The major shortcomings of this approach are that the confidential data of the mobile user may be stored on a cloud server hosted by an attacker, and that cryptographic operations are performed on the mobile device, causing performance penalties.

Yang et al. [30] propose a scheme for mobile devices with limited resources to publicly prove possession of data, which provides privacy and confidentiality. Their scheme involves a TTP for the handling of encoding/decoding, encryption/decryption, signature generation, and verification on behalf of the mobile user.

Zhou and Huang [31] propose a framework that offloads computationally-intensive encryption and decryption operations on a mobile device to the cloud by extending the ciphertext policy attribute-based encryption scheme [32], so that no information about the data contents and security keys is revealed. The main problem with this scheme is that the ciphertext grows linearly with an increasing number of attributes, which hurts real-time performance.

Xiao and Gong [33] propose a collaboration model between mobile and cloud platforms to dynamically generate credentials to protect the mobile user from different types of attacks. The proposed scheme assumes that the cloud is a fully trusted component in the implementation of the proposed security solution. This assumption is too strong especially for public clouds.

Chow et al. [34] propose a policy-based cloud authentication platform for mobile users, which is based on the application of hash functions on data by the mobile platform with a client-generated key and the transfer of the generated information to a data collector in the cloud. The main shortcoming of this approach is that the mobile client has to apply a hash function on frequently generated data to achieve privacy, resulting in high performance overhead.

Bilogrevica et al. [35] propose a solution to preserve the privacy of mobile devices while using the scheduling services on a cloud. The proposed approach uses homomorphic properties of well-known cryptographic systems for secure evaluation of the common availability of mobile users. In the proposed scheme, the mobile device is responsible for encrypting and encoding the messages to ensure indistinguishability, which incurs performance overhead on the device.

Most of the security solutions for mobile-cloud computing listed above focus on the confidentiality of mobile data offloaded to the cloud. Frameworks for protecting the runtime integrity of code offloaded to the cloud are still at a mostly premature state.

3. MOBILE-CLOUD COMPUTING (MCC)

This chapter provides a brief overview of mobile-cloud computing including its definition, advantages, applications, general architecture and challenges. The concepts presented in this chapter prepare the stage for a clear understanding of the subsequent chapters of this dissertation.

3.1 Cloud Computing Overview

3.1.1 Definition

The National Institute of Standards and Technology (NIST) defines **cloud computing** as follows [36]:

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

According to this definition, a cloud computing platform should have the following five essential properties [36]:

1. **On-demand self-service:** A cloud service consumer can provision computing capabilities and use a service automatically with no need of human interaction with the provider.

2. **Broad network access:** Services are made available through a network and can be accessed using standard mechanisms by heterogeneous client platforms.

3. **Resource pooling:** The service provider's resources are shared between multiple consumers using a multi-tenant architecture by pooling. Different physical

and virtual resources are assigned to customers dynamically based on their demands.

4. **Rapid elasticity:** Computing resources can be provisioned and made available to consumers dynamically and automatically by scaling up/down rapidly based on demand.

5. **Measured service:** Cloud systems use a metering capability to optimize resource usage, by monitoring, controlling and reporting usage to the provider and consumers.

3.1.2 Service Models

Cloud service providers offer their services to consumers using one of the following three models:

1. **Software as a Service (SaaS):** This service model allows a consumer to use a cloud service provider's applications running on a cloud infrastructure, which can be accessed through a thin client interface or a program interface. The consumer in this case has no control over the underlying cloud infrastructure, including operating system, network or storage.

2. **Platform as a Service (PaaS):** This service model allows consumers to deploy their own applications on the cloud infrastructure, with access to the programming languages, libraries, services and tools supported by the provider. It is similar to SaaS in the sense that consumers cannot control the infrastructure (underlying network, operationg system etc.), but different in that they can control the applications running on the infrastructure.

3. **Infrastructure as a Service (IaaS)**: This service model provides the customers with the capability to deploy arbitrary software, including operating systems and applications, on the cloud platform. Although this model gives

more control over the computing resources to the customer than the other service models, it still does not provide control over the underlying physical cloud infrastructure. The provisioning of resources in this model is usually achieved using virtualization.

3.1.3 Deployment Models

1. **Private cloud:** In this deployment model, the cloud infrastructure is reserved for use by a single organization consisting of multiple consumers (units).

2. **Community cloud:** This deployment model provisions the cloud infrastructure for use by a specific community of users with shared concerns. The most common application of this model is multiple organizations in an agreement to share resources to achieve a common goal.

3. **Public cloud:** This model provisions the cloud infrastructure for open use by the general public. The deployment is on the premises of the cloud service provider.

4. **Hybrid cloud:** This model is a composition of two or more different cloud infrastructures, which are bound by standard technology enabling data and application portability between them.

3.2 Mobile-Cloud Computing Overview

3.2.1 Definition

Mobile-cloud computing (MCC) refers to an infrastructure where the data storage and processing can happen outside of the mobile device [37]. Mobile-cloud applications move the computing power and data storage away from the mobile devices and into powerful and centralized computing platforms located in clouds, which are then accessed over a wireless connection with those platforms.

3.2.2 Advantages

Mobile-cloud computing offers several advantages over traditional mobile computing, among which are the following [37]:

- **Extending battery lifetime:** Computation offloading migrates large computations and complex processing from resource-limited devices (i.e., mobile devices) to resourceful machines (i.e., servers in clouds). The remote execution of energy-intensive parts of an application can result in significant energy savings. Many mobile applications already take advantage of task migration and remote processing to extend battery lifetime.

- **Improving data storage capacity and processing power:** MCC enables mobile users to store/access large data on the cloud, therefore overcoming the storage capacity constraints on mobile devices. One of the most significant advantages of MCC is that it helps reduce the makespan for computationally-intensive applications, making real-time complex computation possible for mobile applications.

- **Improving reliability and availability:** Keeping data and applications in clouds provides increased reliability and availability due to replication and backups by cloud service providers.

- **Dynamic resource provisioning:** MCC provides dynamic, on-demand provisioning of computational resources to mobile users on a fine-grained, self-service basis. The resources do not need to be reserved in advance, which provides ease-of-use and economic advantage.

- **Scalability:** MCC allows mobile applications to scale to meet unpredictable user demands, by providing access to powerful servers in the cloud and using the elasticity feature of cloud platforms.

- **Ease of integration of services from multiple providers:** MCC allows for the easy integrations of multiple services from different providers through the cloud to meet changing demands of mobile users.

3.2.3 Applications

MCC has been is in use by many mobile applications in the market since its inception. Below is a list of the major fields of mobile applications, which benefit from the power of MCC [37], among many others.

- **Augmented reality:** Augmented reality (a view of the real-world supplemented by computer-generated input such as sound, video, graphics, location data etc.) applications have been increasingly popular since mobile computing devices became commonplace. Augmented reality applications involve computationally-intensive tasks such as video/image processing, vision or voice-based interaction etc., which mobile devices face several limitations in achieving on their own in real-time. Use of cloud platforms for such computationally-intensive tasks enables augmented reality on mobile devices with limited resources [38].

- **Security and emergency applications:** Homeland security and emergency applications are increasingly exploiting the power of mobile computing to ensure widespread presence, which they need for real-time response to catastrophic events. Most of these applications require computationally-intensive tasks such as searching through a large collection of data in real-time, fast image processing for face recognition etc., which need powerful servers for computation. Cloud computing comes to the rescue for these safety-critical applications, providing the needed resources.

- **Mobile healthcare:** Mobile healthcare applications, having the goal of overcoming the limitations of traditional medical treatment, provide easy access to resources such as health records, comprehensive health monitoring services and intelligent emergency management, among other features. In order to deliver the best service experience, these applications rely on cloud resources for the storage and processing of the wealth of data they gather.

- **Mobile gaming:** The gaming industry is ever-growing, creating games with advanced graphics requiring complex calculations. Mobile gaming is another field that greatly benefits from offloading computation to the cloud, as shown by Cuervo et al. [11]. The speed of cloud resources in processing the game state provides a satisfactory gaming experience, while also prolonging the battery life of the mobile device.

- **Mobile commerce:** Mobile commerce applications provide support for achieving commercial tasks requiring mobility, such as mobile transactions, mobile payments etc. Because these applications face challenges such as low network bandwidth and complex security operations, they can greatly benefit from the use of cloud computing platforms to complete their tasks in real-time.

- **Assistive technologies:** Assistive technologies for people with various disabilities are among the applications to get the greatest benefit from MCC. These technologies require portability as an essential feature, which makes lightweight mobile devices a good match for that purpose, but they also involve highly complex operations such as image processing (for visually-impaired people) and speech recognition (for hearing-impaired people), which rely on more powerful computing resources.

3.2.4 Architecture

Figure 3.1 shows the general architecture for MCC [37]. Mobile devices are connected to mobile networks via wireless access points or base transceiver stations (BTS). Service requests from mobile devices are delivered to a cloud through the

Internet using these mobile networks. Access controllers to the cloud receive service requests and route them to appropriate servers in the cloud. Upon receipt of requests, cloud servers process the requests and present the mobile user with the corresponding service through the established communication link. The services could be provided in any of the cloud service models mentioned in section 3.1.2.

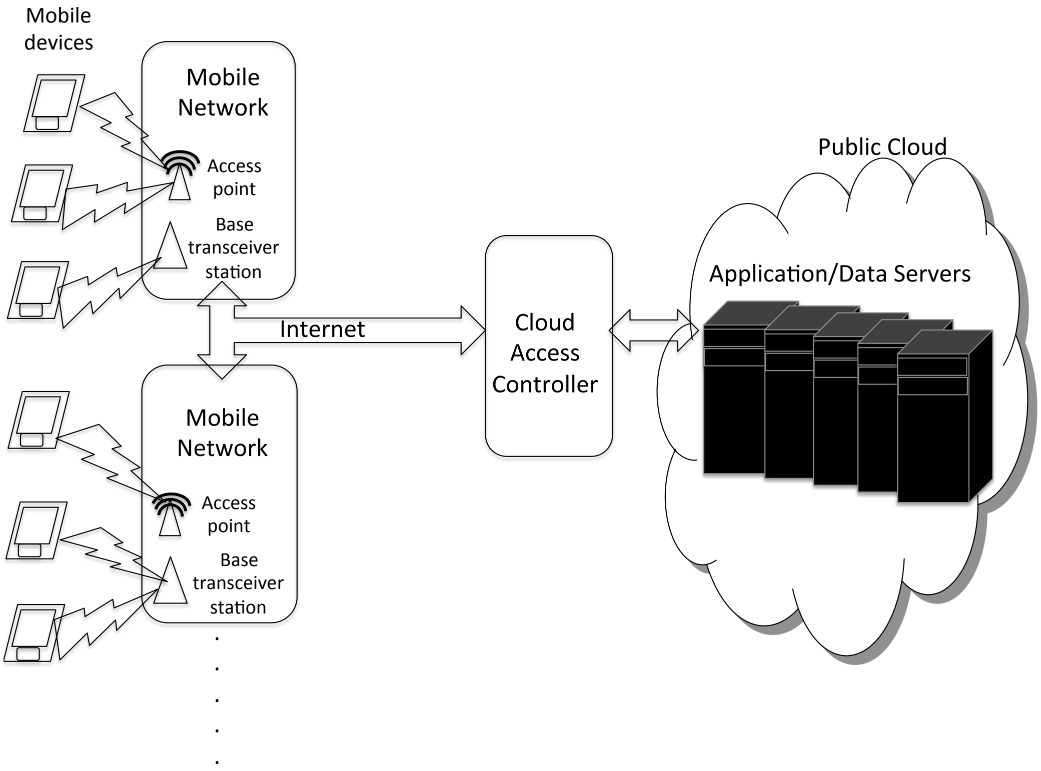


Fig. 3.1.: General MCC architecture.

3.3 Dynamic MCC Challenges

Although MCC is a promising tool to overcome the several challenges faced by mobile devices with limited computational power, it faces challenges too due to the highly dynamic environments it has to operate in. The major challenges that MCC faces include interoperability and standardization issues, security issues arising from the sharing of data and computation with untrusted platforms, and efficient and effective computation partitioning and offloading in real-time. The subsections below provide details about each of these challenges.

3.3.1 Interoperability and Standardization

Interoperability is the ability of a system to work with (offer services to and accept services from) different systems without substantial effort. In the context of MCC, interoperability refers to the ability of a mobile-cloud system to work with different cloud service providers, different cloud architectures and different mobile devices. Because of the strict reliance of MCC performance on immediate access to cloud resources, ability to function with multiple cloud providers is essential, as the resource availability of different providers could fluctuate. To ensure interoperability, it is important for an MCC model to use standard communication interfaces and be operable in different cloud platforms. Operability in different cloud platforms is contingent upon minimal reliance on very specific infrastructure requirements, such as non-standard software, libraries, devices etc.

While a high degree of interoperability is desirable for MCC, as for any other system, it could prove difficult to achieve it due to other constraints such as performance, need of a special runtime environment with specific software etc., which is a major challenge for MCC.

3.3.2 Efficient and Effective Computation Partititioning/Offloading

One of the greatest challenges MCC faces in highly dynamic environments is efficient and effective partitioning and offloading of mobile computation. Here, efficiency of computation partitioning/offloading refers to the overhead incurred by the computation partitioning/offloading mechanism in terms of factors like response time and energy. The effectiveness of the computation partitioning/offloading mechanism determines whether the execution of the resulting computation partitions will provide improved performance over a monolithic execution of the application on the mobile device. Providing a general solution to these challenges is a difficult problem due to the following reasons:

- Variable bandwidth: Mobile environments face high variability in available network bandwidth. A fluctuating bandwidth requires frequent network monitoring to make the best offloading decisions, which comes with the price of monitoring costs in terms of time and energy.
- Mobile service availability: The highly dynamic nature of mobile environments expose mobile users to frequent disconnections due to network traffic congestion, network failures and low signal strength problems. A computation partitioning/offloading model that completely disregards the possibility of such failures will not be able to make optimal decisions.
- Difficulty of runtime conditions estimation: In highly dynamic environments like the mobile and cloud platforms in MCC, it is difficult to estimate runtime conditions of the platform on which the application is running. The environment could be shared with other processes, which could incur unexpected performance penalties. Especially in the lack of specific quality of service (QoS) guarantees by the cloud platform, the makespan of the offloaded computation could be highly variable, which is an important factor for computation offloading decisions.

3.3.3 Security

While MCC is promising to overcome the computational limitations of mobile devices, offloading mobile computation to the cloud entails security risks associated with handing sensitive data and code over to an untrusted platform, including but not limited to the following:

- Lack of control on resources and multi-tenancy of different users' applications on the same physical machine make cloud platforms vulnerable to attacks.

- In addition to privacy issues, programs running in the cloud are prone to tampering with code, data, execution flow and communication, as well as masquerading attacks.
- Mobile code can navigate through multiple platforms before returning to the origin, giving rise to the end-to-end security problem, which involves decreasing control with every further hop in the chain of platforms.

In order to provide complete security, the application should ensure all communication with/execution on the cloud platforms are trusted. Achieving ultimate flexibility and performance in mobile-cloud computing is contingent upon the availability of a secure computing framework capable of dynamic decision making with regards to the execution location of different program partitions.

3.4 Chapter Summary

In this chapter, we provided a brief overview of the general concepts of mobile-cloud computing and the challenges involved in implementing a dynamic computation offloading framework for this computing paradigm. The list of dynamic MCC challenges presented in this chapter serves as a guide for the essential features of the computation offloading framework we propose in this work.

4. AN AUTONOMOUS AGENTS-BASED COMPUTATION OFFLOADING FRAMEWORK FOR MCC

This chapter provides an overview of mobile (autonomous) agents, describes the details of our proposed computation offloading framework for MCC based on mobile agents, and provides a performance evaluation of the proposed framework for three real-world mobile applications.

4.1 Mobile Agents Overview

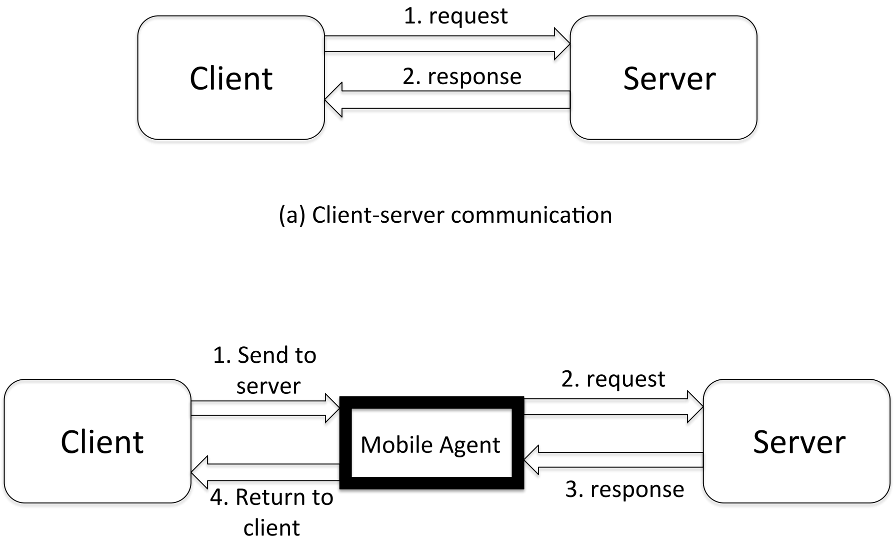
4.1.1 What is a Mobile Agent?

A mobile agent is a software program with mobility, which can be sent out from a computer into a network and roam among the computer nodes in the network [39]. It can be executed on those computers to finish its task on behalf of its owner. When an application using mobile agents needs to request a service from a remote server, it gathers the required information and passes it to the agent's execution environment. At some point during its lifetime, the agent executes an instruction for migration, which results in the following sequence of events [39, 40]:

1. The current agent process is suspended or a new child process is created.
2. The suspended (or new child) process is converted into a message including all of its state information (process state, stack, heap, external references). This message is addressed to the destination where execution will continue.
3. The message is routed to the destination server, which delivers it to the server's agent execution environment.

4. The message is reconstituted to an executable and the associated process is dispatched.

5. Execution continues with the next instruction in the agent program. Upon completion of the agent program at the remote server, the agent might terminate its execution, become resident at the server or migrate back to the originating client or another server.



(a) Client-server communication

(b) Mobile agent communication

Fig. 4.1.: Mobile agent vs. client-server communication.

4.1.2 Advantages of Agent-based Computing

Mobile agents provide several advantages over other distributed computing models such as the client-server model, especially in the case of mobile clients. Figure 4.1 illustrates the main differences between agent-based computing and client-server computing. The main advantages of mobile agent computing, making them effective tools for mobile-cloud computing are the following [39]:

- Agents can provide better support for mobile clients: Mobile devices such as smartphones are intermittently connected to a network, especially in the case of roaming in areas with low GSM/CDMA coverage. Even when the cellular signal is strong, data communication costs set a limit to users' willingness to use applications involving continuous transfer of data from/to a remote server. Yet, due to the limited processing capacities of mobile devices, applications running on them need to offload computationally-intensive operations to more powerful servers. This makes mobile agents, which are based on asynchronous interaction with servers, ideal for mobile client systems.

- Agents facilitate real-time interaction with server: In the case of high network latency, executing a program on a server which provides resources the program needs access to, will be faster than transferring information over the communication link with that server. This makes agent computing a better fit for satisfying the real-time requirements of applications than traditional approaches that maintain constant communication between the client and the server.

- Agent-based queries/transactions can be more robust: Mobile agent computing offers greater reliability than the client-server paradigm due to two main reasons: Asynchronous communication provides reliable transport between the client and the server without requiring reliable communication, and the mobile agent is capable of dealing with a server's unavailability to provide service (in which case it would be routed to a different server with the required service).

- Agent-based transactions avoid the need to preserve process state: The ability of a mobile agent to carry its state around with it relieves the sending host of the need to preserve state, which could otherwise add considerable burden to the client.

- An agent-based application module is capable of moving across different cloud machine instances transparently, without requiring management by its caller module. This feature makes it capable of migrating to a different location for reasons including poor performance on its current execution location or a hostile (attack-prone) runtime environment.

- An autonomous application module can be equipped with techniques to check self-integrity before and after execution, independent of the host platform. This is important for ensuring tamper-proof results from an untrusted platform, which is a major problem for the large-scale adoption of cloud computing today.

Note that hereafter, we will be using the phrases *mobile agent* and *autonomous agent* interchangeably, as the agents in our proposed framework are both mobile and autonomous.

4.1.3 JADE Agent Development Framework

JADE (Java Agent Development Environment) is a popular software framework to develop agent applications in compliance with the FIPA (Foundation for Intelligent Physical Agents) specifications for interoperable intelligent multi-agent systems [41]. JADE is written purely in Java, which makes object-oriented programming in heterogeneous environments possible with features including object serialization and remote method invocation (RMI). All agent communication in JADE is achieved through message passing. Each agent execution environment is a Java Virtual Machine (JVM) and communication between different virtual machines (VMs) as well as event signaling within a single VM is achieved with Java Remote Method Invocation (RMI). Figure 4.2 demonstrates the distributed JADE architecture [42].

The Agent Management System (AMS) controls access to the platform and it is responsible for the authentication of resident agents as well as control of registrations. The Directory Facilitator (DF) is an agent that provides a yellow pages service to the agent platform. Each agent container is a multi-threaded execution environment

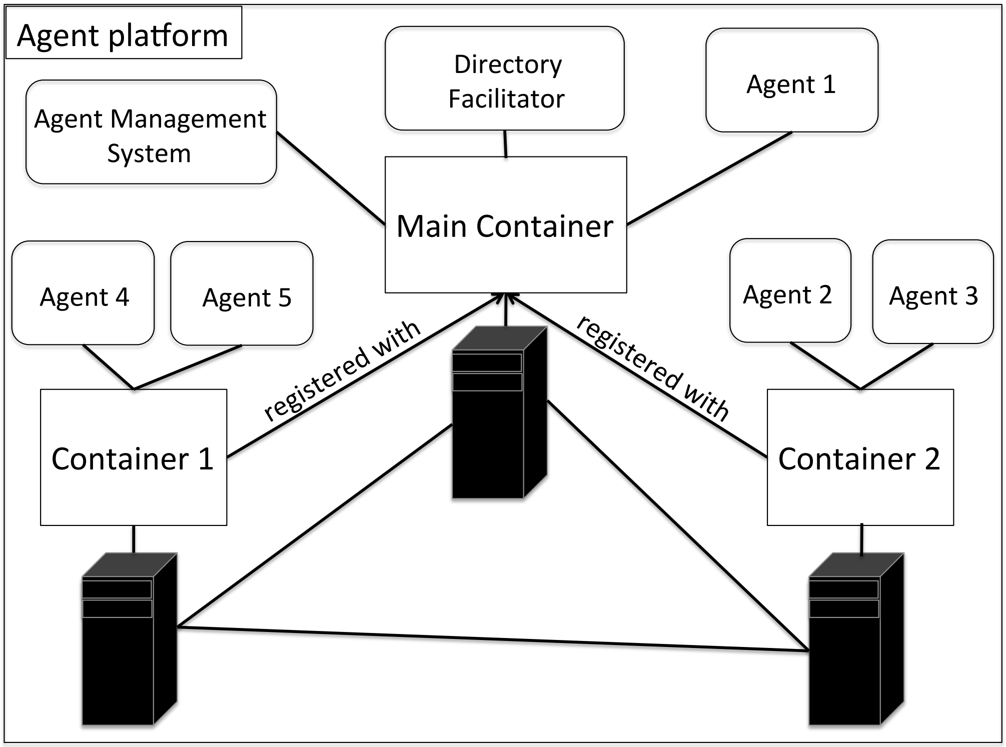


Fig. 4.2.: JADE distributed architecture.

composed of one thread for every agent plus system threads spawned by RMI runtime system for message dispatching. Each agent container is an RMI server object that locally manages a set of agents. It controls the life cycle of agents by creating, suspending, resuming and killing them. Besides, it deals with all the communication aspects by dispatching incoming messages and routing them according to the destination field [41].

Every agent in the JADE framework is composed of a single execution thread, and tasks (code) to be executed by agents are implemented as *Behaviour* objects. In order to have an agent execute a specific task, the task should be implemented as a subclass of the Behaviour class and added to the agent task list using the relevant application programming interface (API).

We chose JADE as the mobile agent development platform for this work due to its prevalence in mobile agent computing and support for multiple platforms including Android OS [43].

4.2 Proposed Computation Offloading Framework: AAMCC

Figure 4.3 shows a high level view of the proposed computation framework, and figure 4.4 shows the framework in action for a mobile application consisting of three modules. Each mobile application in the proposed framework consists of a set of agent-based application modules that are offloadable to the cloud for execution ($P2$ and $P3$ in figure 4.4), in addition to a set of native application components that are always executed on the device due to constraints such as accessing native sensors of the device, modifying native state or providing the user interface of the application ($P1$ in figure 4.4). Partitioning of the application into these two types of components is performed statically before the installation of the application on the mobile device. During the offline application partitioning process, any program partition that is not computationally intensive should not be set as an offloadable component even if it does not have to be pinned to the device, as this would incur additional runtime processing overhead for partitions that would likely never be offloaded. Therefore we advocate the use of programmer help for identifying the offloadable partitions much like the approach taken in MAUI [11].

When a mobile application is launched, the *execution manager* contacts the *cloud directory service* to get a list of available machine instances in the cloud and selects the instance(s) with the highest communication link speed with the mobile device as well as the highest computing power. After this step, an execution plan containing offloading decisions for the agent-based modules is created by the execution manager. If the execution plan requires offloading a particular application module, a bridge is formed between the caller of that module and the *cloud host* selected by the execution manager, through which the offloaded module migrates to the container in the host,

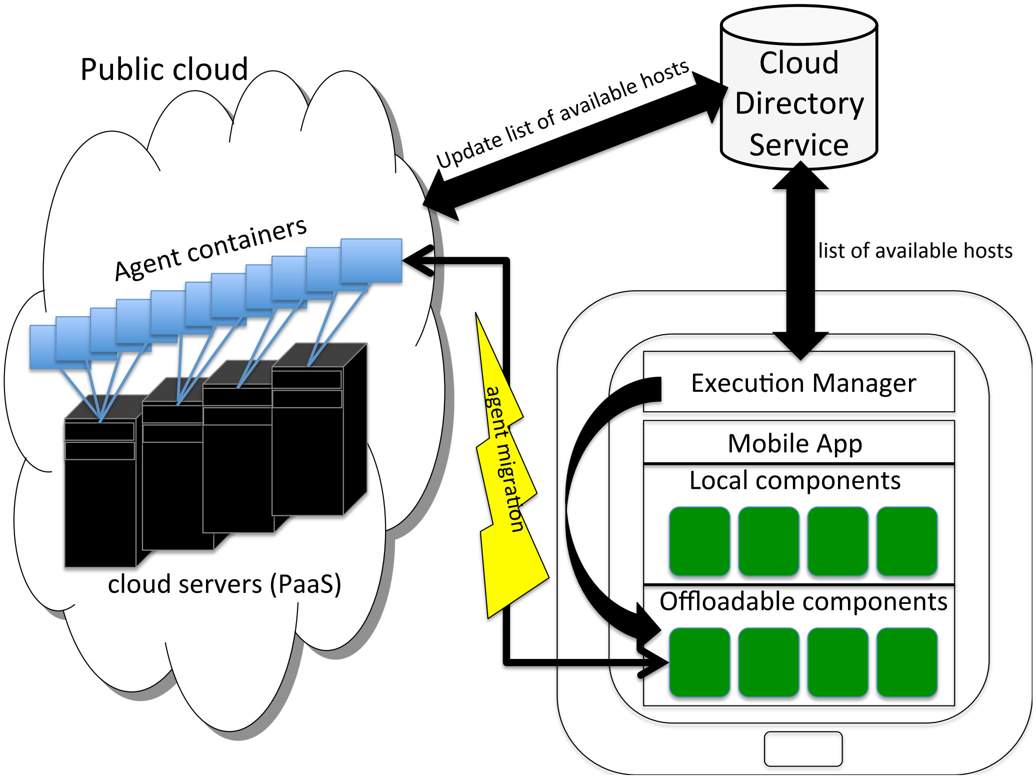


Fig. 4.3.: High level view of the proposed computation framework.

carrying along its input parameters. Upon migration, the module starts executing and communicates its output data to the caller through the same bridge. In the case that the communication between the mobile platform and the cloud platform where the offloaded agent resides is disrupted before the mobile platform receives the computation result, the agent remains on the cloud platform until the mobile platform reconnects to request the result. The ability to reconnect to the agent provides high reliability for the framework, obviating the need to execute the offloaded task from scratch, which could incur high performance penalties.

4.2.1 Proposed Framework Components

The four main components of the proposed framework are described below.

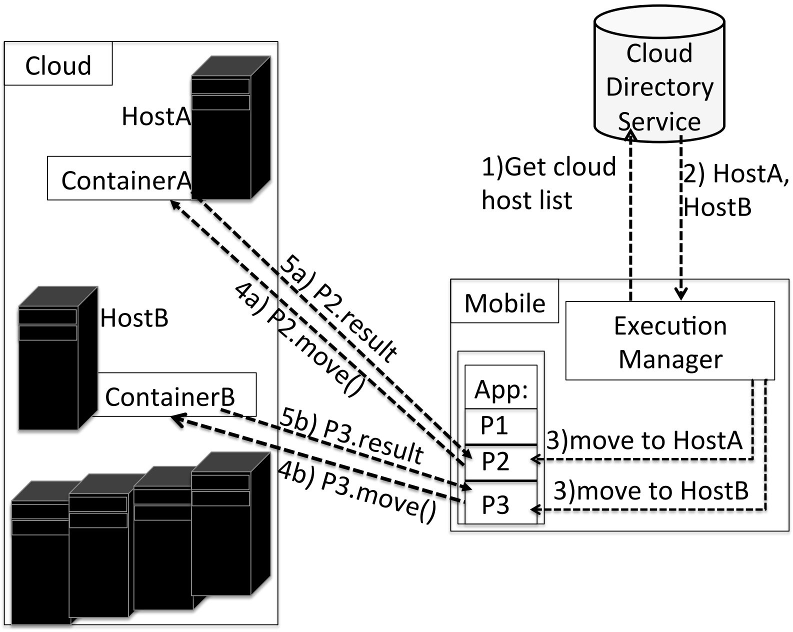


Fig. 4.4.: Proposed computation framework in action for sample mobile application.

Autonomous Application Modules

An autonomous application module is a chunk of application code packed in a mobile agent that is executable in a cloud host. Agent-based application modules provide great advantages over existing mobile-cloud program offloading techniques discussed in section 2 due to their autonomous computing capabilities. Autonomy of these application modules is particularly useful in the context of mobile-cloud computing due to the capability of transparently moving between cloud hosts without requiring management by their caller and self-cloning in different cloud hosts, which can help boost performance in the case of changing runtime conditions in the cloud.

The transformation from a regular application module to the corresponding agent-based module is achieved through a *behaviour* as described in section 4.2.2. In the current framework, application modules have either class-level or method-level granularity, but the same arguments would apply to finer granularity application components as well (such as part of a method).

Execution Manager

This component is a service running on the mobile device, responsible for probing the network for bandwidth and latency measurement and making the decision regarding the execution platform of the different agent-based application partitions. In order to decide where to execute the application partitions, the execution manager contacts the *cloud directory service* to get a list of cloud hosts (virtual machine instances) that are available for use, and estimates the bandwidth of the connection with the hosts. It then runs an optimization algorithm to find the partitioning of the offloadable application modules between the mobile device and the selected host(s) that minimizes the total execution time. Application modules are offloaded/run on the device according to the results of the partitioning decisions.

Cloud Directory Service

The *cloud directory service* is a web service that maintains an up-to-date database of virtual machine instances (cloud hosts) available for use in the cloud. Along with the public IP addresses of the machine instances, information regarding the architecture of the instances, physical regions of the instances and performance records of the most recent interactions with the instances can be provided by this service.

Cloud Hosts

Virtual machine instances (VMIs) in the cloud provide a runtime environment for agent-based application partitions. They provide platform as a service (PaaS), rather than software as a service (SaaS), and the only requirement they need to satisfy is to provide an isolated container (such as a Java Virtual Machine) for each offloaded partition to execute in. In the prototype framework implemented, this component corresponds to JADE agent containers running on Amazon EC2 virtual machine instances [44].

4.2.2 Implementation of Agent Behaviours

As mentioned in section 4.1.3, each application partition needs to be implemented as an agent behaviour in the proposed computation offloading framework. The transformation between a regular Java method and an agent behaviour is quite simple. Listing 4.1 shows part of the code for a Java class *Queens* (for the NQueens puzzle described in section 4.3.3), and listing 4.2 shows the corresponding agent behaviour class. While all local and global variables of the original class remain the same in the agent class, method parameters are saved in a special storage structure called the *Datastore*, and the main method call in the class constructor is moved inside a special method called *action*.

```java
public class Queens {
  int[] x;
  public static ArrayList<int[]> solutions;
  public Queens(int numQueens) {
    x = new int[numQueens];
    solutions = new ArrayList<int[]>();
    callplaceNQueens();
  }
  public void callplaceNQueens() {
    placeNqueens(0, x.length);
  }
  ...
}
```

Listing 4.1: Sample mobile application module code

```
public class QueensBehaviour extends OneShotBehaviour {
  int[] x;
  public static ArrayList<int[]> solutions;
  int numQueens;
  public QueensBehaviour() {
    DataStore ds = getDataStore();
    numQueens = (int) ds.get(”numQueens”);
  }
  public void action() {
    x = new int[numQueens];
    solutions = new ArrayList<int[]>();
    callplaceNQueens();
  }
  public void callplaceNQueens() {
    placeNqueens(0, x.length);
  }
  ...
}
```

Listing 4.2: Agent behaviour implementation

The transformation from a regular mobile application module to an agent-based module is performed statically before application installation, as the overhead of transformation should be avoided at runtime to achieve optimal performance. For every agent-based application module, there is a corresponding regular application module. Whether the regular module or the agent-based module will be executed is decided by the execution manager at application runtime.

4.3 Experiments with Proposed Computation Offloading Framework

4.3.1 Experiments with Face Recognition

The performance of the proposed mobile-cloud computing framework was evaluated with two face recognition applications, which given the picture of a person, identify the most similar face to it in a set of pictures. Both applications are based on the face recognition program available at [45]. The autonomous application modules for each application were implemented as JADE [41] mobile agents and simulations were performed using virtual machine instances with JADE containers in the Amazon EC2 [44] as the servers for code offloading. All sets of experiments were run using an Android emulator over a period of 12 hours to capture possible variations in network conditions and the results reported are the averages for 75 runs.

Face Recognition with Online Data

This face recognition application assumes that the set of pictures to compare against are available online (as in the case of pictures of people in the online social network of the mobile user). Figure 4.5 provides a comparison of executing the application wholly on the mobile device vs. using the proposed offloading framework, for different number of pictures to compare against. We see that the proposed model achieves significantly higher performance than device-only execution, completing the recognition task in as much as 49 times shorter time for the biggest number of pictures.

Face Recognition with Local Data

This application assumes that the set of pictures to compare against is only locally available on the mobile device, i.e. if the application component processing the pictures is offloaded, the data for each picture is sent with it too. Figure 4.6 provides a comparison of executing the application wholly on the mobile device vs. using the

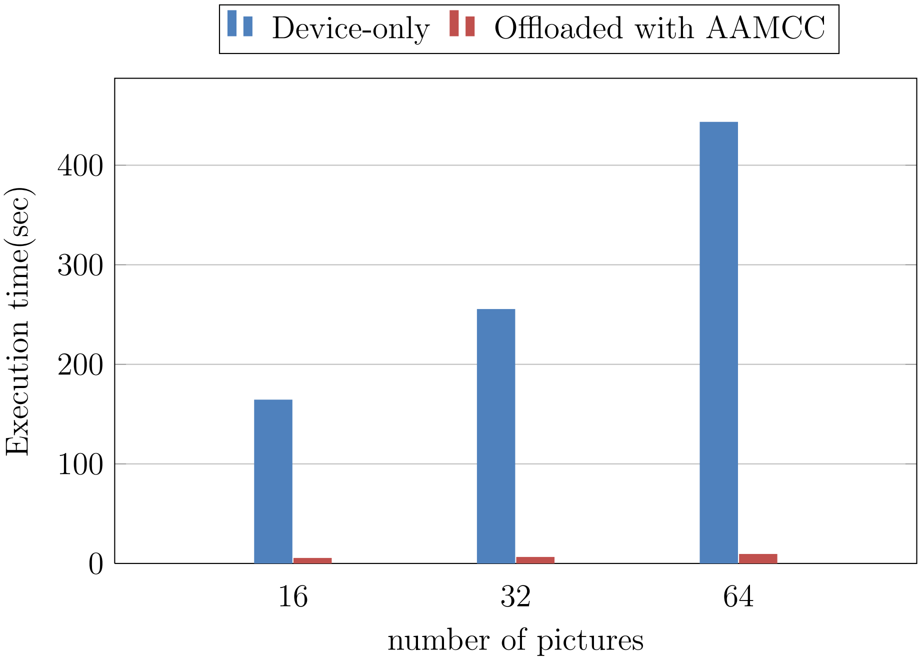


Fig. 4.5.: Execution time vs. number of pictures to compare against for face recognition application with online data.

proposed offloading framework, for different number of pictures to compare against. We see that the proposed model achieves a significant 13 times shorter execution time than the device-only approach for all picture set sizes.

4.3.2 Experiments with Sudoku

Sudoku is a logic-based number-placement puzzle, where the objective is to fill a 9x9 grid with digits (1-9) such that each column, each row, and each of the nine 3x3 subgrids composing the grid contain all digits from 1 to 9, i.e. the same digit cannot appear in the same row, column or in any of the 3x3 subgrids of the board. A puzzle with 17 filled cells has a unique solution. In the Sudoku application used in the experiments, the Sudoku solver component finds and stores all possible solutions for a Sudoku puzzle with a given initial configuration using a recursive algorithm. Although the problem is solvable in moderate time when 17 or more cells are filled in the

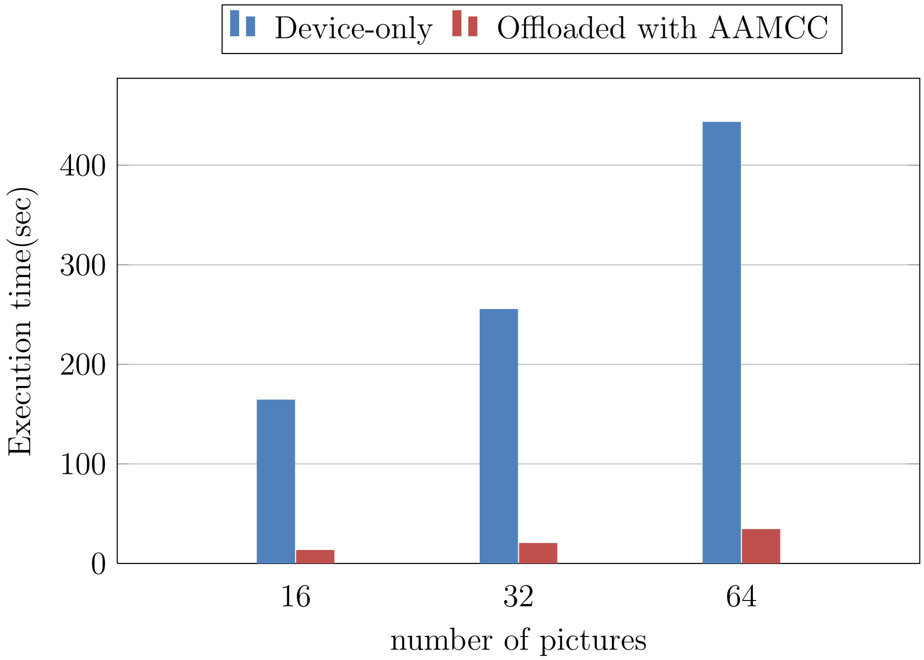


Fig. 4.6.: Execution time vs. number of pictures to compare against for face recognition application with local-only data.

initial configuration, it becomes increasingly computation-intensive with a decreasing number of initially filled cells.

The experiments were performed using a Motorola Atrix 4G [46], where the cloud platform was a medium VMI in the east1-a region of EC2. Each experiment was repeated 10 times and we report the average numbers over the runs. Figure 4.7 provides a comparison for device-only vs. offloaded execution time of the Sudoku solver component for different numbers of initially filled cells. We observe that while the execution time for on-device and offloaded execution are very close to each other for puzzle configurations of down to 18 initially filled cells, offloaded execution achieves 34 times better performance than on-device execution when the initial board has 14 filled cells.

We also measured the average energy consumption of the application on the mobile device using the PowerTutor tool developed by Gordon et al. [47]. Figure 4.8 shows the total (Wi-Fi + CPU) energy consumption on the mobile device for device-only vs.

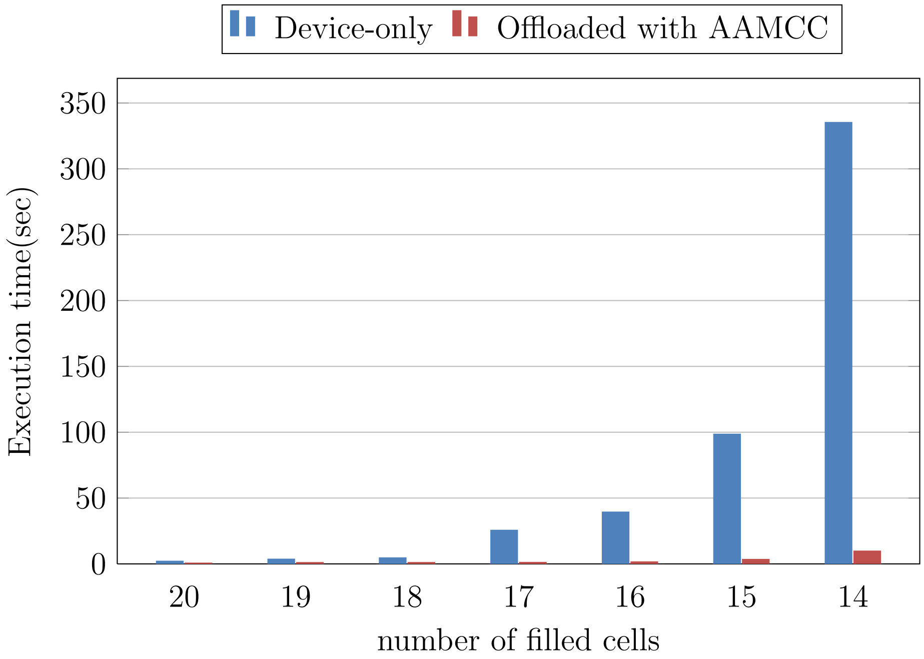


Fig. 4.7.: Sudoku solver execution time for device-only vs. offloaded execution.

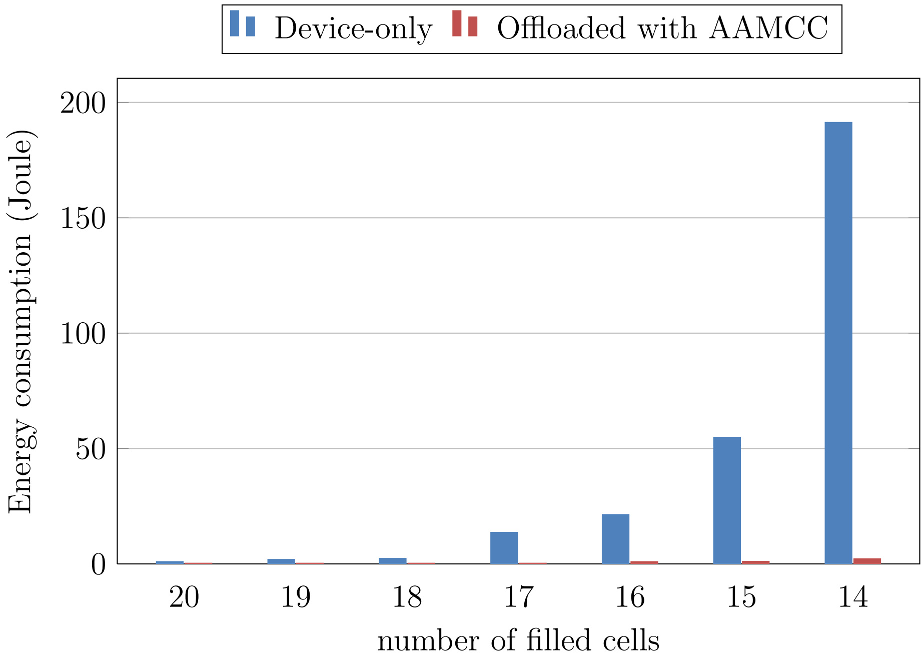


Fig. 4.8.: Sudoku solver energy consumption for device-only vs. offloaded execution.

offloaded execution of the Sudoku solver. We see that offloaded execution achieves significantly higher performance than device-only execution in terms of energy consumption too, consuming 87 times less energy for the lowest number of initially filled cells.

Figure 4.9 shows the distribution of energy consumption over Wi-Fi and CPU utilization in the case of offloaded execution. When the number of initially filled cells is between 17-20, Wi-Fi energy consumption is relatively stable, as the application receives a single (unique) puzzle solution. For fewer initially filled cells, the data received from the cloud is bigger, carrying all possible puzzle solutions for the given initial state. We also see a spike in CPU utilization on the device for those cases: This can be explained by the need to allocate space for the received data on the mobile device.

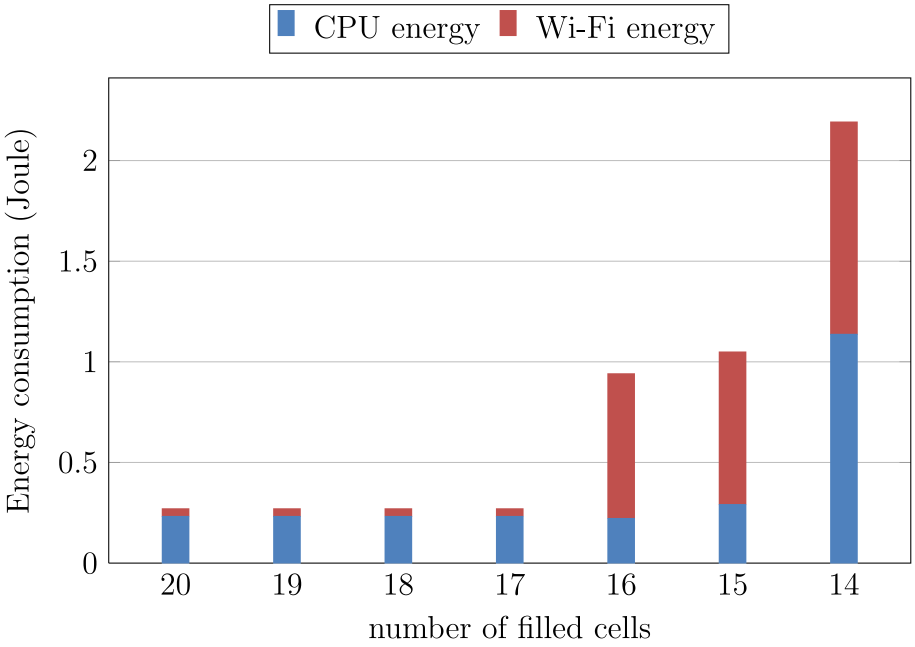


Fig. 4.9.: Sudoku solver energy components for offloaded execution.

4.3.3 Experiments with NQueens Puzzle

The NQueens puzzle is the problem of placing n chess queens on an n x n chessboard such that no two queens can attack each other, i.e. no two queens share the same row, column or diagonal. Solutions to the problem exist for all natural numbers n except for 2 and 3. In the following sets of experiments, we used an NQueens puzzle application, which either finds and stores all possible solutions to the puzzle for a given n (Case 1), or finds all possible solutions, but only stores the number of solutions (Case 2). The experiments below used the same setup as those in section 4.3.2.

Case 1: Returning All Possible Solutions for the Puzzle

In this version of the NQueens application, the puzzle solver component finds all possible solutions for the puzzle using a backtracking (recursive) algorithm and stores them in memory.

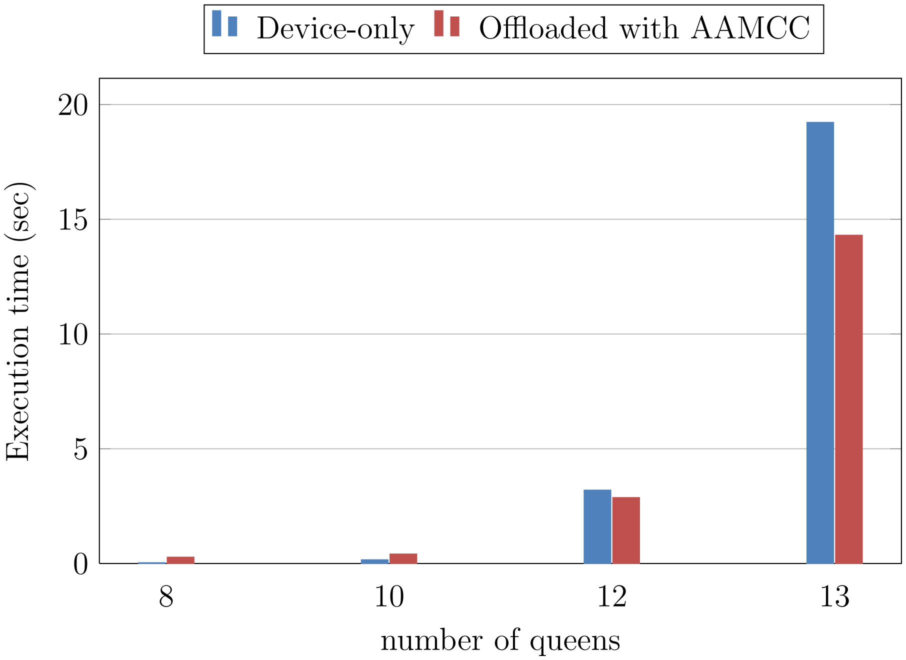


Fig. 4.10.: NQueens solver (all solutions) execution time for device-only vs. offloaded execution.

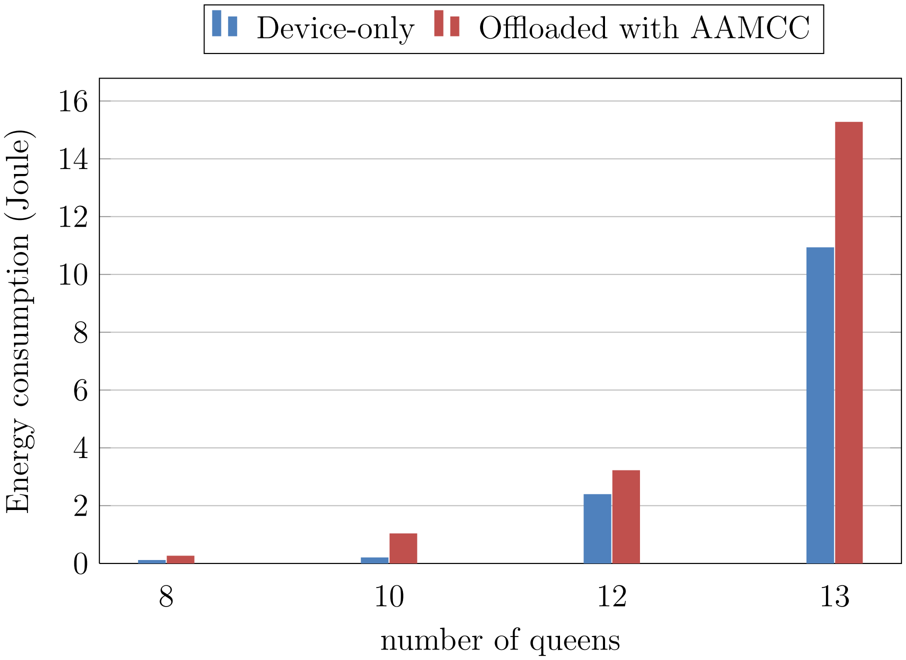


Fig. 4.11.: NQueens solver (all solutions) energy consumption for device-only vs. offloaded execution.

Figure 4.10 shows a comparison of the device-only vs. offloaded execution time for this version of the NQueens puzzle application for different number of queens. Note that the mobile device cannot handle more than 13 queens due to memory limitations. As seen in the figure, while the execution times are close to each other for up to 12 queens, offloading performs about 25% better for 13 queens.

Figure 4.11 shows the total energy consumption on the mobile device for this set of experiments. We observe that offloading always consumes more energy than the device-only approach, with the difference getting sharper with increasing number of queens. For 13 queens, there are 73712 solutions to the puzzle, resulting in both a high data transfer cost and a high memory allocation cost in terms of energy, whereas 8 queens with only 92 solutions has moderate energy consumption.

Figure 4.12 shows the distribution of energy consumption over the CPU and Wi-Fi components for offloaded execution. We see that the percentage of the two energy components are quite close to each other for every case.

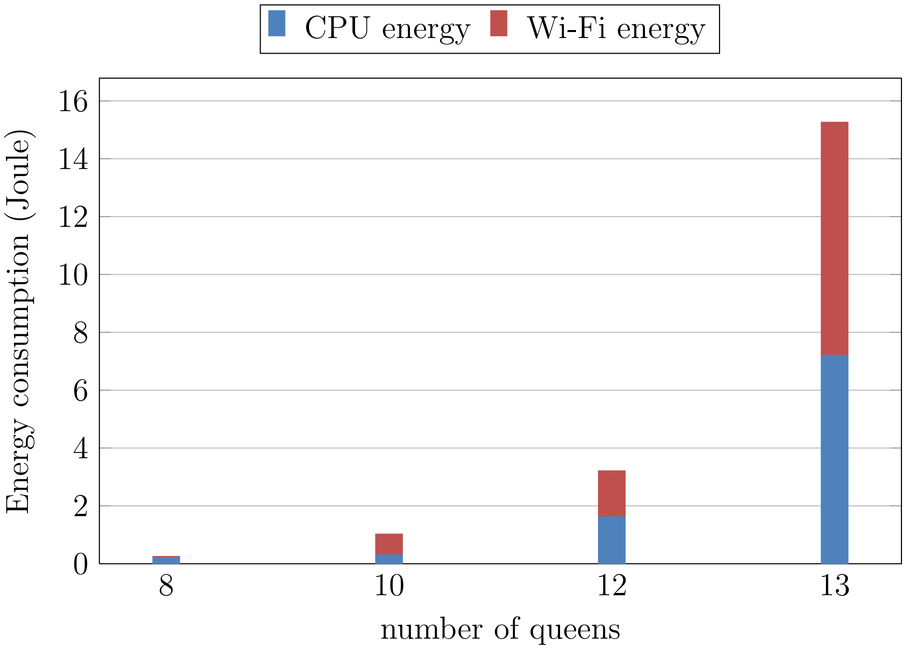


Fig. 4.12.: NQueens solver (all solutions) energy components for offloaded execution.

Case 2: Returning the Number of Solutions for the Puzzle

In this version of the NQueens application, the puzzle solver component still finds all solutions to the puzzle using the same backtracking algorithm as in section 4.3.3, but only returns the number of solutions (i.e. puzzles are solved in place, not requiring memory allocation for each solution). Figure 4.13 provides a comparison of the device-only vs. offloaded execution times for this version of the application for different number of queens. While the execution times are close to each other for up to 12 queens, offloaded execution achieves significantly better performance than on-device execution for more than 13 queens. Actually, it outperforms on-device execution by about 15 times when the number of queens is 15.

Figure 4.14 shows a comparison of the total energy consumption on the mobile device for device-only vs. offloaded execution. The energy consumption for offloading is significantly lower than for device-only execution when the number of queens is more than 13. For 15 queens, offloading consumes about 1600 times less energy.

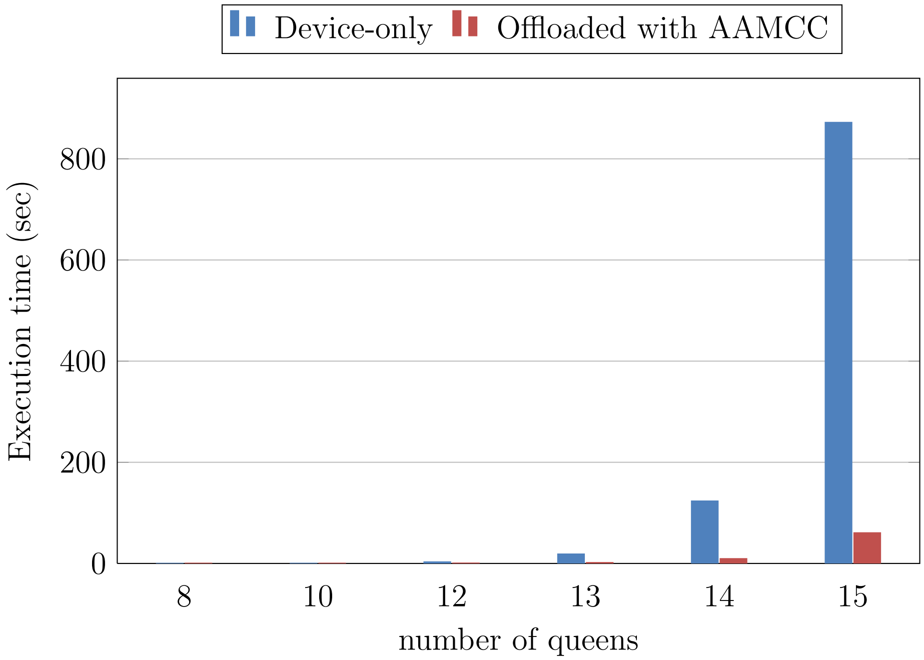


Fig. 4.13.: NQueens solver (number of solutions) execution time for device-only vs. offloaded execution.

The distribution of energy consumption over Wi-Fi and CPU utilization for offloaded execution is seen in Figure 4.15. As expected, Wi-Fi energy consumption is very low, as the response data contains only a single integer.

4.4 Chapter Summary

In this chapter we proposed a dynamic computation offloading framework for MCC based on autonomous application modules offloaded to the cloud in the form of mobile agents. The proposed framework takes advantage of the high performance and reliability characteristics of mobile agents to achieve computationally-intensive tasks of mobile applications, with an easy-to-adopt interface and minimal cloud infrastructure requirements. Experiments with three real-world applications show that the proposed framework achieves significantly higher makespan performance than on-device execution of computationally-intensive tasks, suggesting that it is a promising tool for high-performance dynamic MCC.

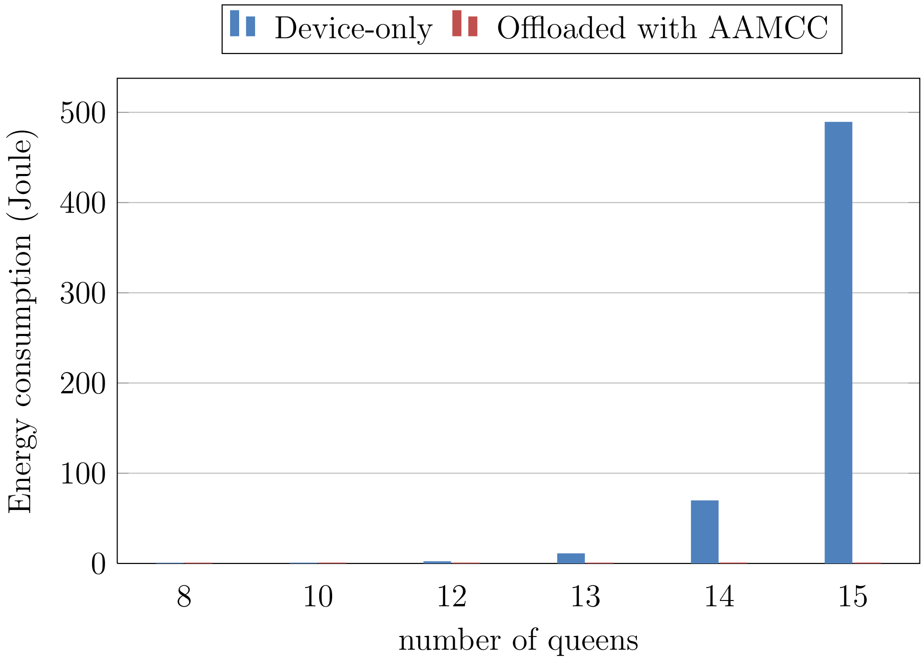


Fig. 4.14.: NQueens solver (number of solutions) energy consumption for device-only vs. offloaded execution.

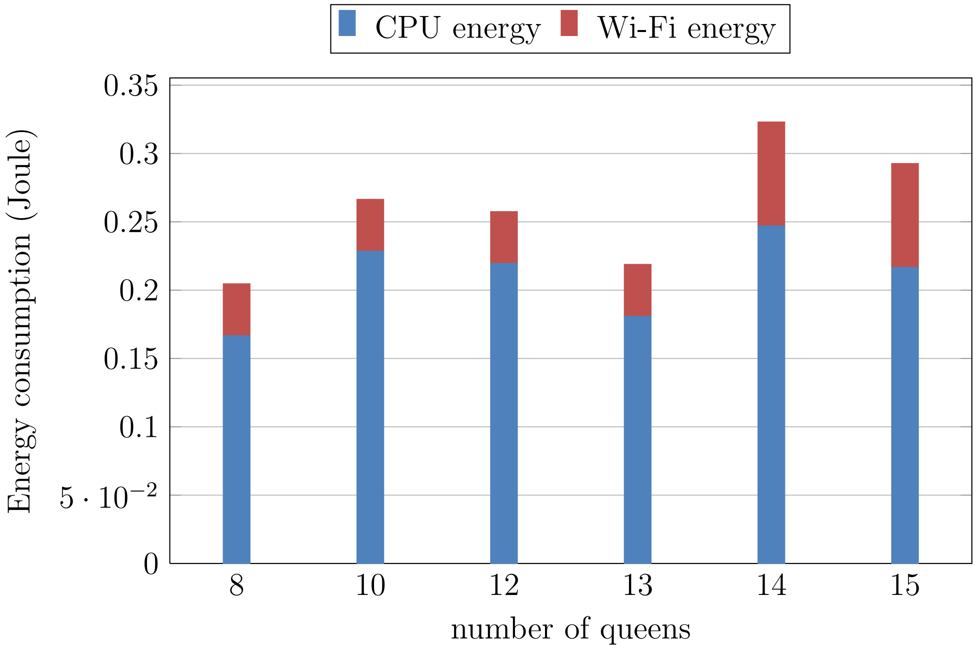


Fig. 4.15.: NQueens solver (number of solutions) energy components for offloaded execution.

5. TAMPER-RESISTANT EXECUTION IN THE CLOUD

One of the most important security issues in MCC is the violation of the integrity of the code and/or data sent to the cloud. Any tampering with code or data in the cloud will lead to inaccurate information/results being received by the mobile platform, which defeats the whole purpose of offloading computation to the cloud. In this chapter, we focus on the design and evaluation of a tamper-resistance mechanism for AAMCC, addressing both the general security challenges for MCC and the security issues peculiar to mobile agents.

5.1 Tamper-Resistance Overview

Tamperproofing a program is ensuring that it *executes as intended*, even when there is an adversary who tries to disrupt, monitor or change the execution. An attacker typically modifies a program with the intent to force it to execute in a way to serve malicious purposes. The attacker can achieve this in one of the following ways [48]:

- Removing code from and/or inserting new code into the executable file before execution
- Removing code from and/or inserting new code into the program during execution
- Altering the runtime behavior of the program using externals agents such as emulators, debuggers, or a hostile operating system

Cloud computing platforms are especially vulnerable to tampering due to the multi-tenancy of applications from different parties on the same physical machine. In a public cloud, it is hard to control the intentions and actions of the different users of a platform. A security case study of the Amazon EC2 by Ristenpart et al. [49] has revealed the vulnerabilities introduced by virtualization, which is the main technology most cloud computing providers rely on to provide shared access to the same physical infrastructure. According to the study, it is possible for an attacker to identify where a particular virtual machine is likely to reside and launch virtual machines on the same physical machine as the target. Tamperproofing code executing in the cloud is especially important to mitigate the effects of such targeted attacks.

Tamperproofing techniques follow the detection of tamper (checking whether data and/or code has been modified) by an appropriate response mechanism, where the execution stops with a report of tamper or the code/data is corrected dynamically to continue execution.

The most common methods to check for the existence of tampering are the following [48]:

- *Checking code*: This method checks that the code executed has no difference from the original code, usually using hashing techniques to compare the original hash value of the code with that at runtime. A collision-resistant hashing algorithm will ensure that any tampering will be detected with high probability when using this method.

- *Checking results*: This method checks the result of some computation in the program to ensure that it falls in a certain range or matches an exact preset value, in order to reason about the existence of tamper in the code/data. Although this method is more efficient than checking the code itself, it could miss cases of tamper that modifies parts of code without affecting the expected result, i.e. this method will not be sufficient to detect tamper when used alone, especially in cases of large software.

- *Checking the environment*: This method, being one of the most difficult to implement, monitors the runtime environment to check its trustworthiness. The checks performed during the monitoring process can include whether the program is running under emulation, whether a debugger is attached to the process, whether the operating system is at the proper patch level, etc. It is especially difficult to check for such conditions in a foreign execution platform such as a public cloud, where the program has no control over most hardware and software resources of the platform, and the responses to the checks performed cannot be fully trusted.

5.2 Mobile Agent Security Issues

While mobile agents offer many advantages in distributed computing, they are prone to many security risks due to the loss of control over their execution in untrusted environments. Security issues associated with the use of mobile agents can be broadly categorized under three headings: Attacks on mobile agents by malicious agent hosts, attacks on mobile agents by other mobile agents and attacks on hosts by mobile agents. In this chapter we consider the first two types of attacks since our objective is to ensure confidentiality and integrity of mobile code executing in the cloud and not to protect cloud resources against malicious mobile agents. The attacks of these first two types can be summarized as follows [40]:

- Leak out/modify mobile agent's code: As the mobile agent's code is executed on the host platform, the host can read the instructions in the code to comprehend the agent's behavior and by accessing the memory location where the code resides, it can modify the agent's code.

- Leak out/modify mobile agent's data: If a malicious host discovers the location of sensitive data such as cryptographic keys that belong to the mobile agent, it can modify the data or use the data for malicious purposes, such as leaking it to outside parties.

- Leak out/modify mobile agent's execution flow: By discovering the mobile agent's physical address of the counter, code and data, the malicious host can predict the set of instructions to be executed next by the agent. With this information, it can change the execution flow to achieve a malicious goal.

- Masquerading: With this scheme, a malicious host can pretend to be the destination platform of a mobile agent, which would allow it to get the sensitive data of the mobile agent and even hurt the original platform's reputation.

- Leak out/Modify the interaction between a mobile agent and other parties: A malicious host might eavesdrop on the communication between a mobile agent and other parties to extract sensitive data (man-in-the-middle attack). This way, it can even modify the contents of the communication and expose itself as part of the interaction.

In the context of mobile-cloud computing, security mechanisms for protecting mobile agents should satisfy the constraints of:

- Real-time response under intermittent network connection

- Keeping communication costs with the mobile platform at minimum

- Incurring limited computation overhead

In order to satisfy the first two constraints, it is important to keep the involvement of the mobile platform in the tamper-resistance framework at a minimum, which can be achieved by augmenting the agents with self-protection capability, as described in section 5.3 below.

5.3 Self-Protecting Agents

The tamper-resistance mechanism we propose for AAMCC is based on the idea of self-protecting mobile agents, which are capable of detecting tampering with their code and data dynamically and taking appropriate actions to report the tamper and move away from the compromised platform. The use of self-protecting agents for tamper-resistance in AAMCC is inspired by two main ideas: Active bundles for self-protecting data and dynamic software integrity verification constructs similar to the software guards proposed by Chang and Atallah [50]. Below we provide details of the these two data/software protection techniques.

5.3.1 Active Bundles

An *active bundle* is a data protection mechanism that encapsulates sensitive data with metadata and a virtual machine. This mechanism takes an information-centric approach for data protection, where data is protected from within instead of from outside as proposed by [51] (a similar approach was taken by Ametller et al. [52] for protecting mobile agents). Active bundles have been successfully used for entity-centric identity management in the cloud [53], and secure data dissemination in a peer-to-peer network of unmanned aerial vehicles [54], which allows for selective data disclosure based on the trustworthiness of the platform the data are sent to.

The structure of an active bundle is shown in figure 5.1. Sensitive data constitutes content to be protected from privacy violations, data leaks and unauthorized dissemination. Metadata describes the active bundle and its privacy policies. The metadata includes (but is not limited to) the following components: (a) provenance metadata; (b) integrity check metadata; (c) access control metadata; (d) dissemination control metadata; (e) life duration value. The policy enforcer (virtual machine) manages and controls the program enclosed in a bundle. The main VM functions include (a) enforcing bundle access control policies through apoptosis (self-destruction) or data filtering (b) enforcing bundle dissemination policies; and (c) validating bundle integrity. What follows is a description of the lifecycle of an active bundle (AB) [51].

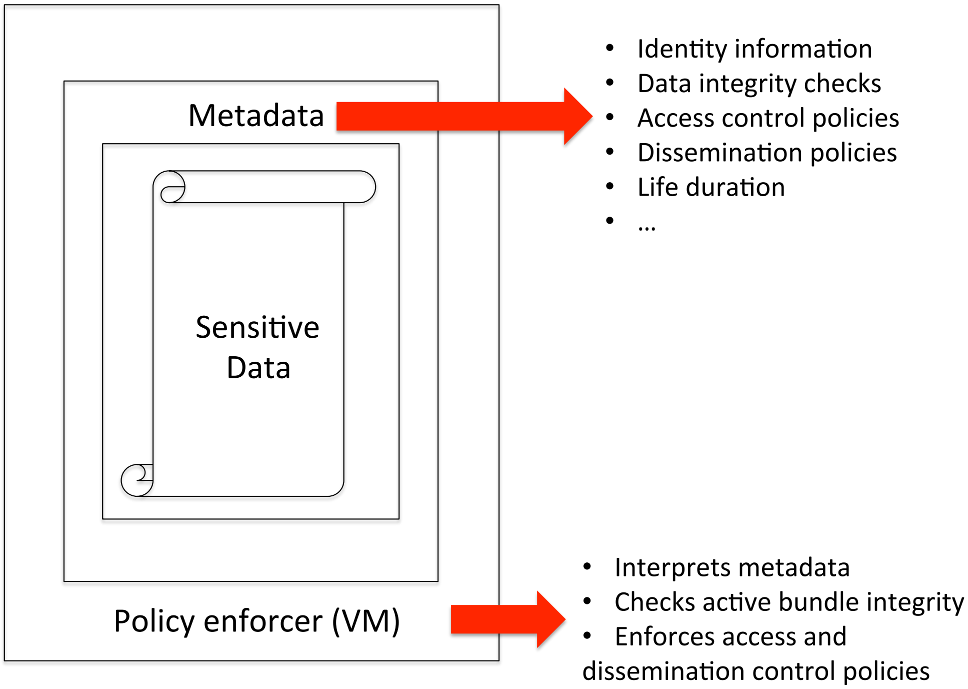


Fig. 5.1.: Active bundle structure.

Initialization of an AB: An owner of sensitive data constructs an AB by putting together data, metadata, and a virtual machine. After this stage, the AB becomes an active entity capable of self-protection.

Building an AB: The steps taken in the process of building an AB are as follows:

1. The AB gets two pairs of public/private keys from a Security Service Agent (SSA), which is a trusted third party (TTP), where the first pair of keys is used for encrypting the AB and the second pair of keys is used for signing/verifying the signature of sensitive data included in the AB.

2. The AB sends a request to SSA asking it to record the AB's security information, which includes its name, a decryption key, and the trust level that a host must satisfy to use the AB. The decryption keys are given only to hosts that are eligible to access the AB.

3. The AB computes a hash value for sensitive data and signs them using the signature key. The signature certifies that sensitive data is from its owner.
4. The AB encrypts sensitive data using the encryption key.

Enabling an AB: After arriving at the destination host, AB enables itself. The steps of the enabling algorithm are as follows:

- Step 1: AB sends a request to SSA asking for the security information on AB and the host's trust level.
- Step 2: AB checks if the host's trust level is lower than the minimal trust level required for AB access. If so, the AB apoptosizes (i.e. completely destroys itself) (executes Step 3); otherwise, it executes Step 4.
- Step 4: AB checks integrity of its sensitive data. It computes the hash value for sensitive data and it verifies AB's signed hash value by comparing it to the computed hash value. If verification fails, AB apoptosizes (Step 5); otherwise, it decrypts the data (Step 6).
- Step 7: AB enforces its privacy policies.
- Step 8: AB provides the output to the host.

As can be deduced from the above description, active bundles provide self protection of sensitive data with the help of an active policy and integrity enforcer, as they move through various foreign platforms. However, once data is shared with a platform, they do not have control over its integrity. This makes them effective tools for data dissemination, but they need to be augmented with other dynamic security mechanisms to prove effective for dissemination and trusted execution of software. One major shortcoming of active bundles for real-time MCC is their reliance on a TTP for their operations including the determination of the trust level of a platform, receiving of decryption keys and audit. Communication with a third party adds increased runtime costs and additional security problems, which is undesirable in MCC.

5.3.2 Introspection

Introspection in the context of software tamper-resistance is the examination of a program or code segment to check that its integrity is preserved. The main method used for code introspection is the augmentation of a program with code that computes a hash value (or a checksum) over a code region and compares it to an expected value [48]. The code snippet in listing 5.1 shows an example of introspection [48] in a C-like syntax. Lines 2-4 in the code setup the starting end ending positions of the code whose integrity is to be checked and lines 5-8 calculates a hash of the code in that region using a special operation $\oplus$. Lines 9-10 act as the response code by taking the action of aborting execution in the case that the calculated hash value does not match the expected value for the region.

```
1            ...
2   start = start_address;
3   end = end_address;
4   h = 0;
5   while ( start < end ) {
6       h = h ⊕ *start;
7       start++;
8   }
9   if (h != expected_value)
10      abort();
11           ...
```

Listing 5.1: Introspection example

Attacks against tamper-resistant software using introspection generally use pattern-matching methods to identify locations in the program that include suspicious code (such as conditionals checking the value of a hash). In order to prevent the response to a tamper detection, these attacks usually modify the response code or replace the hash value with a pre-computed one that would force the program to continue execution. The only way to make sure that tampering will be detected in this case is to ensure the integrity of the tamper-resistance code itself. Even though it may not be possible to ensure perfect resistance against such attacks, especially in the case of large code size, the probability of success of the attacks can be decreased by using the idea of a network of software guards, as described below.

Software Guards

The *software guards* algorithm as proposed by Chang and Atallah [50], is based on the idea that in order to prevent tampering it is not sufficient for guards (introspection code) to check the integrity of the software code: They need to check the integrity of each other as well. The algorithm is based on the following principles:

- The program is divided into code regions, where each region is either user code, a checker guard or a responder guard. A *checker* guard is an execution unit that checks the integrity of a code region by comparing its checksum value to an expected value, whereas a *responder* guard is an execution unit that replaces a tampered code region with the original code for that region.

- A region can be checked by multiple checkers, and a tampered code region can be repaired by multiple responders.

- To provide tamper-resistance, a network of guards is placed in the execution flow graph of the program in a way that satisfies the conditions: (1) every responder guard is inserted into a point that is reached before the execution of the code it guards, (2) every checker guard is inserted at a point in the program during the execution of which the code to be guarded is present in the program image.

- It is possible to create a circular chain of guards such that each guard in the chain protects its neighbor, forming a cycle with no unprotected guards.

The introduction of software guards was a pioneering effort in the field of self-protecting software. The level of protection provided by this approach depends on the number of guards integrated into the program, where a greater number of guards covering a larger part of the code base will provide better protection against attacks. Additional measures to make attacks harder include (1) obfuscating guard code, so that the guard can protect itself in addition to being protected by other guards, and (2) excluding recognizable signatures that the attacker could statically scan for from the guard code. There are two main issues with increasing the resilience of a software program against tampering attacks with these hardening approaches:

1. Code size: Integration of an increased number of guards into the program code results in increased code size. While this may not be an important factor for static software running on a dedicated platform, it could have a significant effect on the performance of mobile code in an MCC system, due to bandwidth limitations and sharing of resources.

2. Runtime performance: The integration of guard code into the program could result in significantly lower performance, especially if the guard code dominates the program. This is especially important in the case of software with real-time requirements, and software that runs on a platform sharing resources with others, as in the case of MCC.

The tamper-resistance approach we propose for AAMCC, described in section 5.6, is inspired by the main idea of software guards that guard code needs tamper detection too. In order to meet the real-time requirements of MCC, we propose a low-overhead tamper detection mechanism as described in section 5.4. One main difference of our approach from the *software guards* approach is that the program does not continue execution in the cloud platform once tampering is detected, as this signifies that the platform is compromised and cannot be trusted further. In order to make the mobile platform aware of the tamper, a form of remote tamper-resistance, as described in section 5.5, is needed.

5.4 Low-Overhead Tamper Detection

The main challenges for achieving dynamic tamper-resistance with self-protecting software in MCC are the following:

- The tamper detection mechanism should involve minimal modification to the software structure, and be transparent to the programmer.
- Appropriate measures should be taken in case of tamper detection in a way that is transparent to the software.
- The runtime performance overhead of tamper detection and response should be minimal.
- Communication costs for detection and reporting of tamper should be minimal.
- The size of the program code/data should not be increased significantly due to the protection mechanism.

Our solution to address these challenges in AAMCC is to use *Aspect-Oriented Programming* (AOP) to integrate integrity verification units, like the software guards described in section 5.3.2, into the autonomous application modules. Section 5.4.1 below provides an overview of AOP.

5.4.1 Aspect-Oriented Programming

Aspect-oriented programming (AOP) is a programming technique that allows for the augmentation of software with cross-cutting concerns, which are behaviors that span multiple, often unrelated, implementation modules [55]. Examples for cross-cutting concerns include security, logging, debugging, synchronization and persistence among others. An *aspect*, as defined by Kiczales et al. [56], is property that affects the performance or semantics of the components of a system, and is not a unit of the system's functional decomposition itself. AOP enables programmers to cleanly separate aspects and software components, so that cross-cutting concerns of a program can be seamlessly integrated into program code, obviating the need to inline the code for the concern in multiple places.

AOP languages are based on five main elements [57, 58]:

- **Join point:** An identifiable point in the execution of a program, such as the execution of a method or the handling of an exception, where enhancements may be added.
- **Advice:** The behavior to be executed at a particular join point. Many AOP frameworks model an advice as an interceptor.
- **Pointcut:** A means of identifying join points (i.e. a predicate that matches join points). An advice is associated with a pointcut expression, and is executed at the join points matched by the pointcut.
- **Introduction:** Declaring additional methods or fields on behalf of a type.
- **Weaving:** A method of attachment of units to a program. This can be done at compile-time, load-time or runtime.

The code snippet in listing 5.2 shows an example of using the AspectJ [59] AOP framework to output the name of each method invoked in the *QueensBehaviour* class from section 4.2.2 and the corresponding makespan.

```
public aspect Profiler {                                                   1
  pointcut methodCalls():                                                  2
    execution(* QueensBehaviour.*(..));                                    3
                                                                           4
  Object around(): methodCalls() {                                         5
    long start = System.currentTimeMillis();                               6
    try {                                                                  7
      return proceed();                                                    8
    }                                                                      9
    finally {                                                             10
      long end = System.currentTimeMillis();                              11
      String method = thisJoinPoint.getSignature().getName();             12
      System.out.println(method + ”\t” + (end - start));                  13
    }                                                                     14
  }                                                                       15
}                                                                         16
```

Listing 5.2: AOP example using AspectJ

Line 2 in the code declares a pointcut *methodCalls*, which is a regular expression to match during the execution of the program. In this case, the pointcut matches all invocations of all methods of the *QueensBehaviour* class. Any point matching this pointcut in the program is a join point. Line 5 declares the code (advice) to be executed upon matching the declared pointcut. The *around* keyword specifies that the enclosed code has parts to be executed before (line 6) and after (lines 10-14) the body of the matched methods, with the body of the method executing in between (as specified by the special method *proceed*). We see that the whole process of adding the makespan profiling functionality is completely transparent to the actual application code.

5.5 Remote Tamper-Resistance

Remote tamper-resistance is a special case of tamper-resistance, where the program to be protected runs on an untrusted platform, but communicates with a master program on the trusted platform, which detects or is made aware of any tampering with the program running on the untrusted platform [48]. This kind of tamper-resistance is especially necessary in the context of MCC, where the mobile platform needs to ensure the integrity of offloaded computation before integrating results/information received from the cloud platform.

One of the main techniques used for remote tamper-resistance is the use of introspection for the program running on the untrusted platform and reporting the hash value of the code to the trusted platform, which can make sure that it matches a pre-computed value. However, this approach is not sufficient to ensure tamper-free execution, as the remote platform can tamper with the hash value before reporting. Two main approaches proposed to mitigate the problem of tampering with the hash value are the following:

1. Making sure that the hash value was computed with the correct code: The Pioneer [60] system proposed by Seshadri et al. relies on a hash function constructed such that, if the remote platform cheats during the computation the computed hash will either be wrong or it will take a longer time than expected to compute it. This approach relies on the knowledge of the exact system properties on the remote platform, such as the CPU model, clock speed, memory latency, memory size etc. to estimate the execution time of the hash function, therefore it is too restrictive for MCC using public clouds, where the mobile platform may not have the knowledge of any of these properties.

2. Continuous code replacement [48]: This approach proposed by Collberg et al. is based on the idea of continuously obfuscating the code executing on the remote platform. The offloaded software in this approach consists of blocks of code, where each block may be modified at runtime based on updates received from

the trusted platform. Frequent updates to the blocks make it difficult for an adversary to analyze the code and modify it to serve malicious purposes. This approach requires constant communication between the platforms, making it unfit for MCC due to bandwidth limitations and the real-time computation requirements.

Even under security models ensuring the integrity of all computation on a foreign platform, remote tamper-resistance relies on secure communication between the trusted and untrusted platforms, as data could still be tampered with in transit. Section 5.5.1 below provides an overview of the secure communication problem in the MCC setting and presents the details of one possible solution to the problem.

5.5.1 Secure Communication with the Cloud

While tamper-resistant execution is one of the requirements for getting trustworthy results from offloaded computation, without a secure communication link between the mobile and cloud platforms, it would not be possible to judge the integrity of the results received. In order to trust the messages sent by an autonomous application module, the mobile platform should be able to verify the following conditions upon message receipt:

1. The message was sent by a legitimate agent (authentication)
2. The message was not modified during transmission (integrity)
3. The content of the message was hidden from the outside world during transmission (confidentiality)
4. The content of the message is in the expected form

One of the most efficient ways to verify the first three conditions all at once is to use *authenticated encryption* for messages sent by mobile agents.

Authenticated Encryption

Authenticated encryption (AE) is a shared-key based transform whose goal is to provide both *privacy* and *authenticity* of the encapsulated data [61, 62]. The encryption process E in an authenticated encryption scheme takes a plaintext p and a shared secret key k and returns a ciphertext c, whereas the decryption process D takes the ciphertext c and the same key k and outputs either the plaintext p or a special symbol $\perp$.

i.e.

$$\begin{aligned} E(k,p) &\rightarrow c \\ D(k,c) &\rightarrow p \cup \perp \end{aligned} \tag{5.1}$$

An output of $\perp$ by D means that the received ciphertext c was invalid or unauthentic. Using AE for a message sent by Alice to Bob, encrypted with their shared key k has two implications:

1. An active adversary (man-in-the-middle) cannot create a ciphertext which would make Bob believe that it was encrypted by Alice, i.e.

   if $D(k,c) \neq \perp \Rightarrow c$ is from someone who knows the shared key k.

2. The message is protected against chosen ciphertext attacks, i.e. the privacy of the message is preserved.

5.6 Proposed Tamper-Resistance Approach

5.6.1 Assumptions and Attack Model

Before delving into the details of the proposed tamper-resistance protocol, we state our assumptions about the environment in which an autonomous application module executes and the common types of attacks such execution platforms are vulnerable to. The proposed tamper-resistance protocol is supposed to operate correctly under the following assumptions:

- The VMI on which the agent is executing resides in a public cloud with no strict security policy enforcements.
- The cloud service provider is trusted, however users of the cloud services may not be trusted.
- A user (attacker) residing on the same physical machine as the agent is capable of modifying data and code at the application layer, but not capable of modifying anything at the hypervisor, host operating system or hardware layers.

Under these assumptions, the autonomous application modules are prone to at least the following kinds of attacks:

- Man-in-the-middle: This attack involves eavesdropping on the communication between the agent in the cloud and the mobile platform, forming independent connections with each and modifying messages exchanged between them [63]. The attack would only be successful if the attacker can impersonate the parties involved, i.e. convince the parties that the messages they receive are coming from the expected party at the other end of the communication link, and not from an attacker.
- Active attacks against code and data integrity: These attacks comprise the most common forms of tampering on a shared execution platform. The attacker could tamper with a program statically before or dynamically during its execution by tampering with the libraries it uses, attaching a debugger to the program and modifying code on-the-fly by replacing values of variables, methods etc.
- Data privacy violations: These attacks involve the revealing of private data in the cloud to an attacker who could use it for malicious purposes. The attacker can extract private information from the agent data during execution or

from the messages exchanged between the agent and the mobile platform by eavesdropping on the communication link.

- Code privacy violations: These attacks involve leaking of proprietary program code to an unauthorized party. It can be achieved by an attacker by making a copy of the code as it is executing with the help of a debugger etc.

5.6.2 Tamper Detection and Reporting Protocol

The approach we propose for tamper-resistant execution of offloaded code in the cloud is based on augmenting the mobile agents sent to a cloud platform with self-protection capability, using the mechanisms detailed in the sections 5.4 and 5.5 above. Integrity checkpoints are distributed throughout the agent code to ensure timely detection of tamper. Upon tamper detection, the agent stops execution, moves to a different platform and either (a) starts execution from the beginning or (b) resumes execution from the last integrity-verified checkpoint. The self-protection method relies on the following two main ideas:

1. Tamper checking guards: As the agent code is executing in the cloud, the integrity of the code is checked using introspection by software guards placed at various *integrity checkpoints*. These guards are also capable of reporting tamper to the mobile platform.

2. Guard tamper tracking code: Each time a software guard checks for tamper, its own hash value is saved in agent data. The agent accumulates the guard hash values in a variable, which forms the key to encrypt the result sent to the mobile platform when the computation is complete. The mobile platform is capable of computing the key value independently, therefore it can decrypt the message received to extract the result from the autonomous application module.

Using an encryption key formed by the dynamic hash values of the software guard code in the authenticated encryption of the result has the following implication: For the correct encryption key to be formed, the guard hash value at each integrity checkpoint needs to be integrated into the key and the integrated value needs to match the true (original) hash value for that guard. If the result received from the mobile agent cannot be decrypted by the mobile platform, there are two possibilities:

1. The message containing the result was modified in transit to the mobile platform.

2. The message was not encrypted with the correct key on the cloud platform.

In either of these conditions, the result cannot be integrated into the program on the mobile platform.

In the proposed security approach, accumulation of guard hash values during agent code execution is a stealthy tamper-resistance mechanism in the sense that it does not involve any checks (conditional expressions) that are prone to detection by pattern matching methods used by attackers. It is important to form the encryption key using all guard hash values, as we are interested in detecting any tamper during the execution of the agent. This incremental formation of the encryption key serves two purposes:

1. The encryption key is not revealed until the end of program execution, which makes it harder for an attacker to dynamically swap the value of the corresponding variable with the original key value to hide tamper.

2. The mobile platform can ensure that all integrity verification blocks in the offloaded module were executed in the case of a correct key value, as the execution of guard hashing code is triggered by the execution of guard code blocks.

As evident from the above description, the strength of the mechanism for detecting tampering with the software guards relies heavily on the strength of the encryption key. In order to create a strong key, allowing for the detection of tamper with high probability, the process of forming the key should satisfy the following conditions:

- All guard hash values should be integrated into the key.

- The individual guard hash values should be formed using a collision-resistant hash function.

- The function forming the encryption key from the individual guard hash values should output uniformly distributed (pseudorandom) results in its output domain.

- The function forming the encryption key should be able to operate incrementally (not necessarily commutative or associative), instead of requiring all input at once. This is because storing the individual hash values throughout the program execution both increases space requirements and makes the hash values prone to modification attacks.

Another condition that needs to be satisfied for the tamper detection mechanism to operate correctly is the ability of the mobile platform to calculate the encryption key independently, as a symmetric-key encryption mechanism is used in the model. This means that integrity checkpoints should be placed in the code in such a way that their execution is independent of the control-flow of the program, i.e. the corresponding guard code is executed no matter which path the program takes based on the runtime conditions.

Let H be a collision-resistant hash function used to calculate the guard hash values. The algorithm to compute the encryption key is provided in Algorithm 1. The result of the offloaded computation is encrypted using an authenticated encryption scheme introduced in section 5.5.1, with the secret key formed using this algorithm.

Algorithm 1: Algorithm for computing AE encryption key

```
input  : $guardCode$: Array of guard code text, $H$: hash function, $F$: guard
         hash accumulation function
output: encryption key for autonomous application module result
$numBlocks \leftarrow size(guardCode)$
$key \leftarrow 0$;
$mostRecentGuard \leftarrow 0$;
while $mostRecentGuard < numBlocks$ do
    $hashOfGuard \leftarrow$ H($guardCode[mostRecentGuard]$);
    $key \leftarrow$ F($key, hashOfGuard)$);
    $mostRecentGuard \leftarrow mostRecentGuard + 1$;
end
```

Figure 5.2 shows the UML activity diagram of the execution lifecycle of a mobile agent in the proposed tamper-resistance framework. The main steps taken by the agent during execution are as follows:

- Step 1: The running *guard hash* value for mobile agent *MA* is reset.
- Step 2: MA is sent to a selected VMI for execution.
- Step 3: MA checks the integrity of its code and data before starting execution on the cloud platform.
- Step 4: If the integrity of MA was preserved during the transfer, MA starts executing the first code block in the application module and goes onto Step 5. Otherwise, a new VMI *V* is selected by inquiring the *cloud directory service*, MA sends a message to the mobile platform including the tamper report and its new VMI address, moves to V and restarts execution from Step 1.
- Step 5: The software guard for the most recently executed code block checks the runtime integrity of the block. If no tamper was detected, the running guard

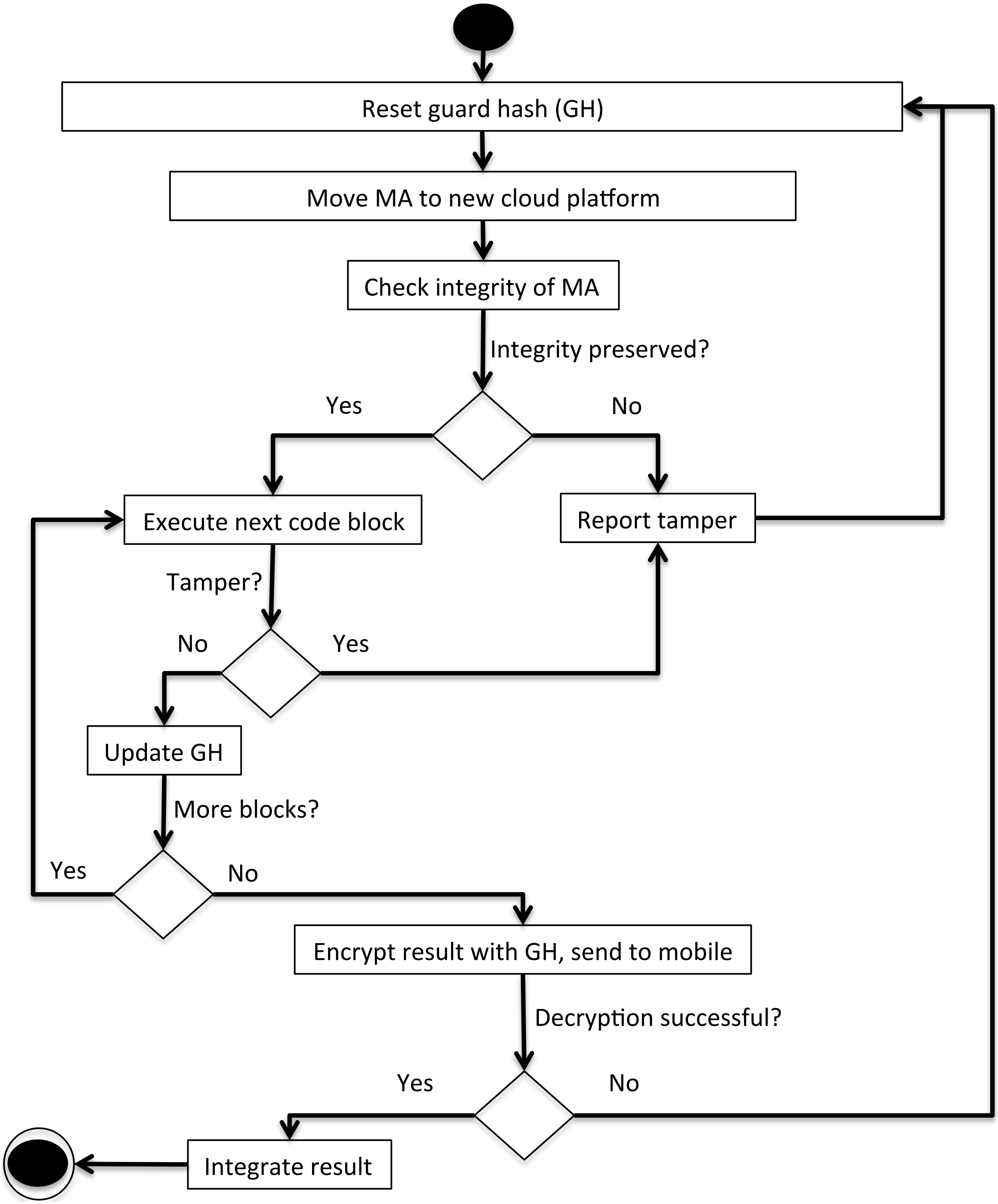


Fig. 5.2.: Activity diagram for proposed tamper-resistance mechanism.

hash value is updated with the hash of the guard code performing the most recent check. Step 5 is repeated until there are no more code blocks to execute.

If tamper was detected by the guard, a new VMI V is selected by inquiring the *cloud directory service*, MA sends a message to the mobile platform including the tamper report and its new VMI address, moves to V and restarts execution from Step 1.

- Step 6: The result of the offloaded computation is encrypted with the guard hash value and sent to the mobile platform.

- Step 7: If the mobile platform is able to successfully decrypt the result message received from MA, it integrates the result into the program. If the decryption of the result message fails on the mobile platform, the whole process of MA's execution is repeated starting with Step 1.

5.6.3 State-based Programming for Resuming Agent Execution after Tamper Detection

The tamper-resistance approach introduced above forces a complete restart of the execution of an autonomous application module every time it moves to a different platform due to tampering. This could introduce significant delay for the receipt of a response on the mobile platform, especially in the case of late tamper detection in program modules of high computational complexity. One approach that could be used to avoid the performance penalty due to tampering is to exploit the *statefulness* of mobile agents to resume execution from the last integrity-preserved point of execution after migration to a new platform.

Figure 5.3 shows a UML activity diagram of the tamper-resistance protocol enhanced with stateful agents capable of resuming computation from the last code block of confirmed integrity. Note that the main structure of the agent lifecycle is the same as that in Figure 5.2, with the major differences being emphasized by rectangles with thicker borders. The main changes to the tamper-resistance algorithm are as follows:

- In addition to saving a guard hash value for every execution block, the agent keeps track of its execution state, i.e. it records the last block of tamper-free execution after each tamper check.

- When tampering is detected and the agent moves to a different platform, it does not start execution from scratch. Instead, it continues execution with the code block during the execution of which tampering was detected.

- The conditions under which the agent restarts execution from scratch (i.e. failed decryption and failed initial integrity check) involve a reset of the agent state in addition to the guard hash.

The introduction of states to the autonomous application modules requires a slight modification of the program code to align integrity checkpoints with agent state updates. An agent program consisting of the sequential code blocks *block*1, *block*2 and *block*3 (where integrity checkpoints are attached) in listing 5.3 is transformed into the stateful program in listing 5.4 to acquire the ability to resume execution with the last integrity-verified checkpoint after migration.

```
public class Stateless {
  public static void main(String [] args){
    block1();
    block2();
    block3();
  }
  ...
}
```

Listing 5.3: Sample stateless program

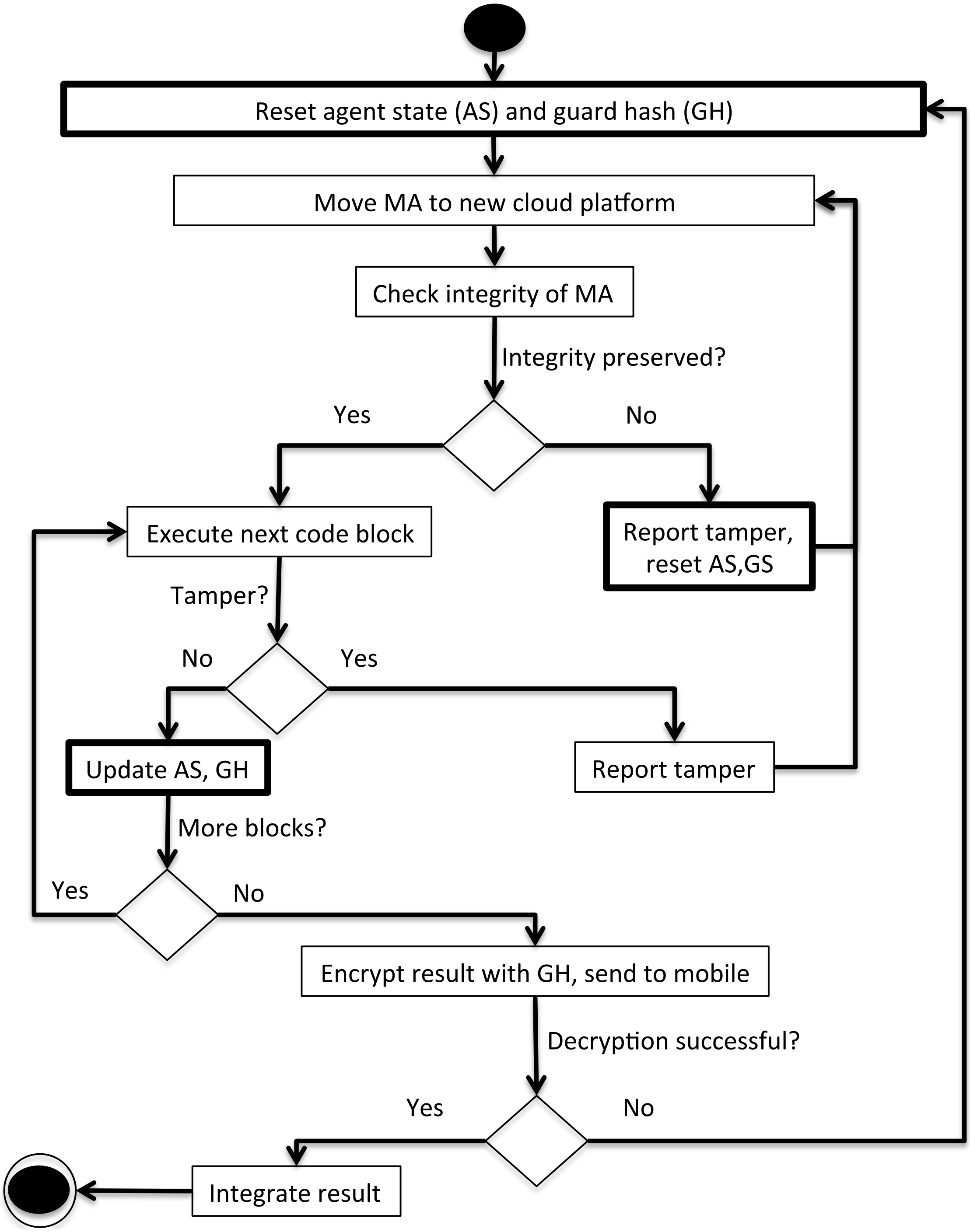


Fig. 5.3.: Activity diagram for proposed tamper-resistance mechanism with states.

```
public class Stateful {
  static int state = 0;
  public static void main(String [] args) {
    if (state == 0) {
      block1();
      state = state + 1;
    }
    else if (state == 1) {
      block2();
      state = state + 1;
    }
    else if (state == 2) {
      block3();
      state = state + 1;
    }
    ...
  }
}
```

Listing 5.4: Stateful program

5.6.4 Implementation Details

In the proof-of-concept implementation of the proposed tamper-resistance framework, integrity checkpoints are inserted around every method call using the *AspectJ* [59] AOP framework. The hash value for each method is calculated on the bytecode for that method using the *SHA-256* secure hash algorithm. The Galois/-Counter (GCM) mode of operation [64] is used for the authenticated encryption of the result by the mobile agent in the cloud platform and the decryption on the mobile platform. GCM is a block cipher mode of operation that uses universal hashing over a binary Galois field to provide authenticated encryption. It can be implemented in hardware to achieve high speeds with low cost and low latency, and software implementations can achieve very high performance by using table-driven field operations. Authenticated encryption using GCM takes three inputs:

- A secret key K
- An initialization vector IV, that can have any number of bits between 1 and 2^{64}. For a fixed value of the key, each IV value must be distinct, but need not have equal lengths.
- A plaintext P, which can have any number of bits between 0 and 2^{39} - 256.

The output is a ciphertext C of the same length as P. The decryption process takes the inputs C, K, and IV and either outputs P or $\perp$, depending on whether there was tampering on the cloud platform or during message transmission.

In the proof-of-concept implementation, the secret key is formed from the bitwise addition of the hash values of guard code, while the initialization vector of length 128 bits is determined using a random number generator on the mobile platform and integrated into agent data before it is sent to a cloud platform for execution.

5.7 Evaluation of Proposed Tamper-Resistance Protocol

5.7.1 Resilience against Attacks

The resilience of the proposed security model against the attacks listed in section 5.6.1 is as follows:

- Man-in-the-middle: The proposed model provides protection against man-in-the-middle attacks with authenticated encryption. An attacker would need to have knowledge of the secret shared key of the mobile platform and the autonomous agent in order to be able to impersonate the agent in a message sent to the mobile platform. Any message encrypted with a key other than the true shared key will not be decryptable by the mobile platform.

- Active attacks against code and data integrity: The proposed model provides protection against active data and code tampering attacks using software guards to check the dynamic integrity of program code and passively tracking the integrity of software guards during execution, any tampering with which is revealed upon attempting decryption of data encrypted with a key whose value is based upon the integrity of guard code. The proposed scheme is resistant against modification of agent code at load-time and hot-swapping of code during runtime. However, any modifications performed at the kernel or the native libraries on the cloud platform, which require higher privileges, will go undetected by this tamper-resistance framework.

  The proposed tamper-resistance framework was tested with (a) tampering with offloded code before execution using a faulty class loader and (b) tampering with offloaded code during runtime using the Javasnoop [65] tool, which allows intercepting methods and altering data during the execution of a Java program, by attaching to the running process. Attacks were implemented against the integrity of both the program code and the guard code. The proposed framework was able to detect all tampering cases.

- Data privacy violations: Although the proposed model provides protection against privacy violations in data communication by using encryption, it does not protect the privacy of data in the cloud once it is revealed by an agent. Protection of sensitive data in AAMCC is provided by avoiding the offloading of such data to the cloud altogether.

- Code privacy violations: The proposed model does not provide protection of code privacy, as the code is run in clear text on the cloud platform. Using encrypted functions would be a way to mitigate this problem, however it could significantly hurt the application performance. Therefore, any code the privacy of which needs protection is kept local on the device in the current computation offloading framework.

5.7.2 Experimental Evaluation

Experiments were performed to evaluate the performance overhead of the proposed tamper-resistance mechanism for the makespan of two of the applications introduced in section 4.3.

Experiments with Face Recognition

The experiments with the face recognition application with local-only data were performed for the case of a picture database of 32 subjects to compare against. An Android emulator was used as the mobile platform, and a medium VMI in the west1-a region of EC2 was used to host the mobile agents. 62 integrity checkpoints were inserted into the agent code. Each experiment was repeated 75 times and we report the average makespans. Figure 5.4 shows the comparison of the average makespan of the face recognition application with code offloading in the case of no tamper-resistance mechanism present and the average makespan of the same application with the proposed tamper-resistance mechanism integrated.

As seen in the figure, the difference between the average makespans for the two cases is always less than 600 ms, which is only about 1.8% of the total execution time. The average makespan for on-device execution for the same application is around 255 sec, therefore the AAMCC with tamper detection still does much better than monolithic execution on a mobile device, and the tamper-resistance overhead is negligible.

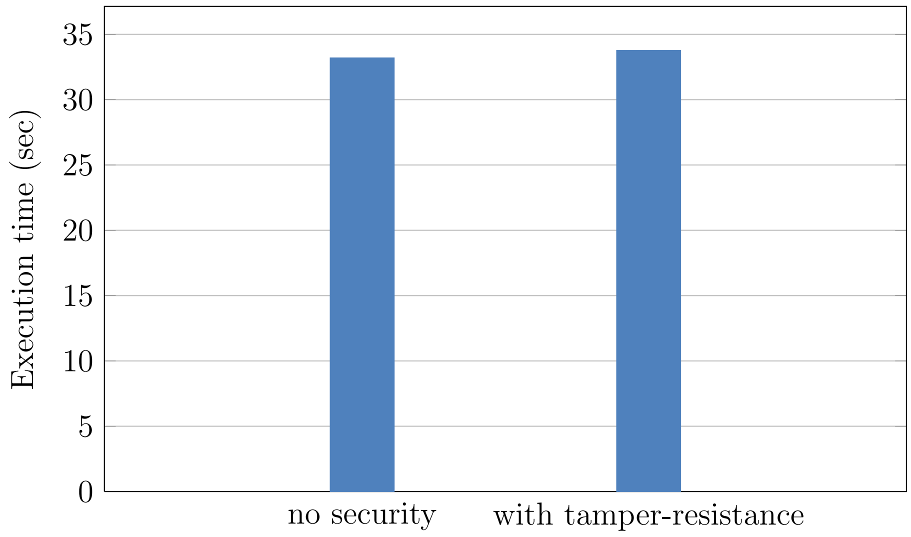


Fig. 5.4.: Comparison of execution times for the face recognition application with vs. without tamper-resistance mechanism.

Experiments with NQueens Puzzle

The experiments with the NQueens application were performed with the version outputting the number of solutions for a puzzle with a particular number of queens. A Motorola Atrix 4G device was used to host the mobile application, whereas a medium VMI in the west1-a region of EC2 was used to host the cloud platform. The connection between the mobile and cloud platforms was provided over a wireless network. 2 integrity checkpoints were inserted into the agent code. Each experiment was repeated 10 times and we report the average makespans. Figure 5.5 shows the comparison of the average makespan of the NQueens application with code offloading in the case of no tamper-resistance mechanism present and the average makespan of the same application with the proposed tamper-resistance mechanism integrated, for different number of queens.

As seen in the figure, the difference between the average makespans for the two cases is always less than 2.2 sec, which is less than 3.6% of the total execution time for the case with 15 queens. The average makespan for on-device execution for the same case is 872 sec, therefore the AAMCC with tamper detection still does much

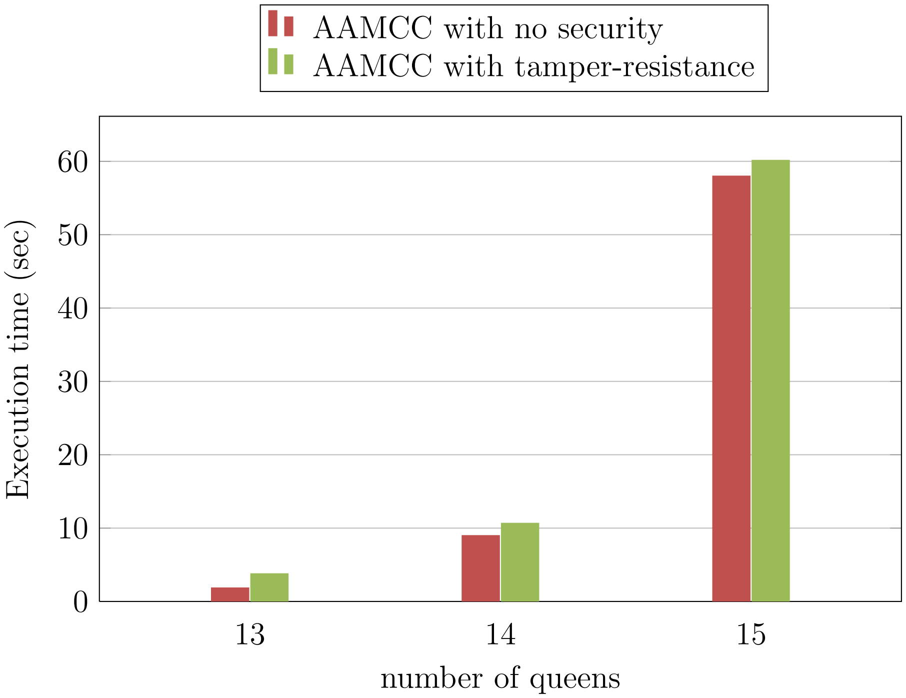


Fig. 5.5.: Comparison of execution times for the NQueens solver application with vs. without tamper-resistance mechanism.

better than execution on the mobile device, and the tamper-resistance overhead is negligible once again.

We also performed experiments where the first VMI to which the NQueens application module was offloaded had a faulty class loader, i.e. the agent code was tampered with at the first hop it was offloaded to. Figure 5.6 shows the comparison of the execution times for (a) device-only execution, (b) offloaded execution with no tampering, and (c) offloaded execution where execution on the first VMI the agent runs on involves tampering, but the next VMI it migrates to after detection of tampering (at the first integrity checkpoint) does not. The execution time for the offloading case is measured as the time between the start of the application and the time when a *correct* result is received from the agent.

Figure 5.6 shows that execution with tampering (involving migration of the agent) involves a fixed extra cost of about 15 seconds over the no-tamper case, due to the migration of the agent, for all numbers of queens. However, even with tampering

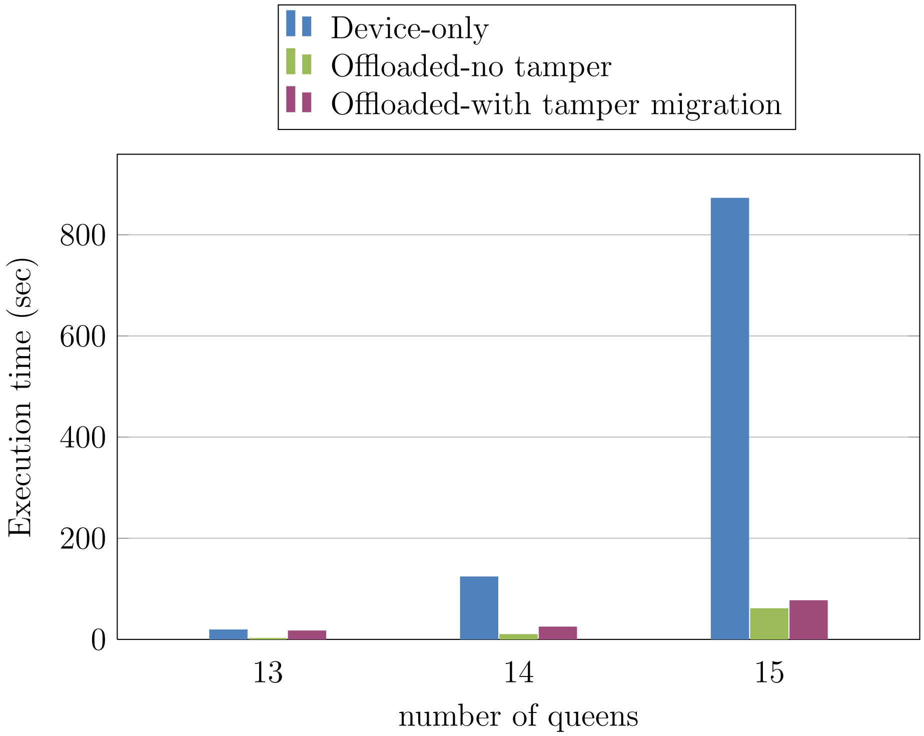


Fig. 5.6.: Comparison of execution times for the NQueens application for (a) device-only execution, (b) offloaded execution with no tampering, and (c) offloaded execution with tampering on first VMI.

involved, AAMCC with tamper-resistance still achieves significantly higher performance than device-only execution.

5.8 Chapter Summary

In this chapter we discussed major security issues with offloading code and data to a public cloud and proposed a tamper-resistance framework for MCC based on self-protecting agents, using various integrity checkpoints of code introspection to detect and report tamper. We also discussed details of a state-based programming model for MCC, where the agent can continue execution from the last integrity-preserving checkpoint after migration to a different platform, instead of starting execution from the beginning. The proposed tamper-resistance framework was shown to incur very low runtime overhead and successfully detect load-time and runtime code tampering attacks. The negligible performance overhead of using self-protecting agents for tamper-resistance proves that mobile agents are effective tools for high-performance, secure MCC.

6. CONTEXT-AWARE MOBILE-CLOUD COMPUTING

This chapter discusses the effects of different types of context on the performance of mobile-cloud computing. We start with motivating experiments using mobile applications under different conditions to demonstrate the importance of context in offloading decision-making. We then propose a computation offloading model for multi-component mobile applications taking into account the network context, and a self-performance evaluation algorithm for agent-based application modules considering the cloud resources context.

6.1 Context in MCC

Context plays a very important role in achieving high quality of service with both mobile and cloud computing, as both computing paradigms face highly dynamic conditions, i.e. highly variable context. Richness and efficient gathering/utilization of contextual information gains even more importance in the case of collaboration between mobile and cloud platforms, as achieving high computational performance and user satisfaction is contingent upon the correct identification of resources to be utilized and various constraints peculiar to mobile computing.

Mobile-cloud context consists of the following elements [66]:

1. User preference: This element of context captures user-specific preference settings for particular mobile applications, and could potentially affect the composition of mobile services.

2. Device context:

    - Device characteristics: This type of context information is related to the inherent features of the mobile device, such as the mobile operating system, processor speed, memory size, etc. It is most useful for determining the expected performance of a mobile application running on the device.

- Energy: This is one of the most utilized and dynamic types of context information in mobile-cloud computing. Energy utilization by different components of a mobile application and the power level of a mobile device during application runtime are strong determinants for computation offloading/service composition decisions in a mobile application.
- Workload: The workload on the mobile device (the usage of device resources by different applications, or by different components of an application) is another highly dynamic element of context affecting application component offloading decisions.

3. Quality of service

   - Data connection type, bandwidth: The available bandwidth between the mobile device and a cloud server and the data connection type are important factors especially for real-time data-processing intensive applications.
   - Cloud resource availability: This contextual clue captures the availability and quality of various resources on specific cloud hosts, and can be used to make decisions regarding the selection of cloud services to achieve the best application performance.

4. Situational context: This context element consists of monitored data and information regarding the mobile user location, time and other data collected by sensors on the device (e.g. accelerometer). It is most useful in personalizing mobile applications to achieve high user satisfaction.

6.2 Effects of Mobile-Cloud Context on Autonomous Agents-based MCC

The subsections below provide an evaluation of the effects of different elements of mobile-cloud context on the performance of AAMCC, by reporting the results of a series of experiments performed using the framework in different contexts on the face recognition and NQueens puzzle applications introduced in section 4.3.

6.2.1 Effect of Network Speed

Figure 6.1 shows the effect of different network speeds on the execution time of the face recognition application with local-only data using AAMCC, when the number of pictures to compare against is fixed at 32. In this case, we observe a significant performance penalty with a slow network, due to the large data transfer involved in offloading. However, the performance of the offloading approach under the slowest network conditions is still significantly better than that of device-only execution.

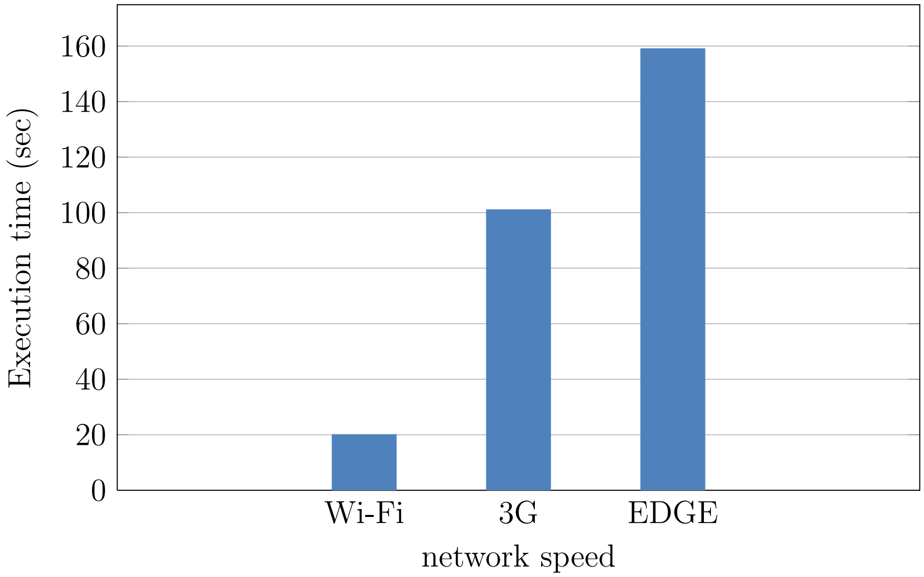


Fig. 6.1.: Execution time vs. network speed for face recognition application with local-only data.

6.2.2 Effect of Multi-Tasking

In this set of experiments, we evaluate the performance of a multi-threaded version of the NQueens puzzle from section 4.3.3, where there are as many threads as the number of queens and each thread of execution finds the number of solutions for the puzzle for a fixed position of the first queen (queen on the first row).

Table 6.1: Single vs. multi-threaded NQueens execution time (in seconds).

number of queens	device-only single thread	device-only multi-thread	offloaded single thread	offloaded multi-thread
8	0.03	0.04	0.31	0.29
10	0.13	0.15	0.30	0.32
12	3.07	2.48	0.61	0.60
13	18.65	10.32	1.77	1.85
14	123.45	63.85	9.46	9.50
15	872.01	447.24	60.60	61.26

Table 6.1 provides a comparison of the execution time performances of the single-threaded vs. multi-threaded versions of the NQueens puzzle solver. Note that offloaded execution uses a single core machine (medium instance) in Amazon EC2. For on-device execution, we observe close to 2-fold speed-up with multi-threading. The number 2 here is attributable to the existence of 2 cores on the mobile device. We also observe that although offloaded execution still performs significantly better than on-device execution, there is no noticeable difference between the single-threaded and multi-threaded versions of offloaded execution, which is best explained by the lack of multiple cores on the cloud host. Table 6.2 shows that the total energy consumption on the mobile device is not affected by multi-threading.

6.2.3 Effect of Cloud VMI Characteristics

We also performed experiments with different machine instance types in Amazon EC2 as the cloud hosts, to see the effect of cloud host architecture on execution time. Specifications for the machine instance types we used in the experiments are shown in Table 6.3.

Table 6.2: Single vs. multi-threaded NQueens energy consumption (in Joules).

number of queens	device-only single thread	device-only multi-thread	offloaded single thread	offloaded multi-thread
8	0.08	0.05	0.20	0.21
10	0.16	0.14	0.27	0.23
12	1.84	2.08	0.26	0.19
13	10.64	10.38	0.22	0.21
14	69.32	70.16	0.32	0.27
15	488.86	493.52	0.29	0.26

Table 6.3: Amazon EC2 instance type specifications [67]

Instance type	memory	number of cores	network performance
Micro	0.615 GB	1	very low
Small	1.7 GB	1	low
Medium	3.75 GB	1	moderate
Large	7.5 GB	2	moderate
2x Large	30 GB	8	high

Figure 6.2 shows the effect of the cloud VMI type on the execution time of the face recognition application with local-only data for different number of pictures to compare against. We observe that the performance of the application degrades by 50% when a small machine instance (1.7 GB memory, 1 core) is used instead of a medium instance (3.75 GB memory, 2 cores).

Figure 6.3 compares execution times for the multi-threaded NQueens puzzle solver for different cloud host types in Amazon EC2. We observe that for 14 and 15 queens, the performance speed-up is commensurate with the number of cores in the host machine, with the 2x large machine instance taking 8 times shorter than the medium instance and 4 times shorter than the large instance for the same number of queens.

These results suggest that resource provisioning in the cloud has a significant effect on the performance of mobile-cloud applications, and that offloading decisions should be made based on the characteristics of the available resources.

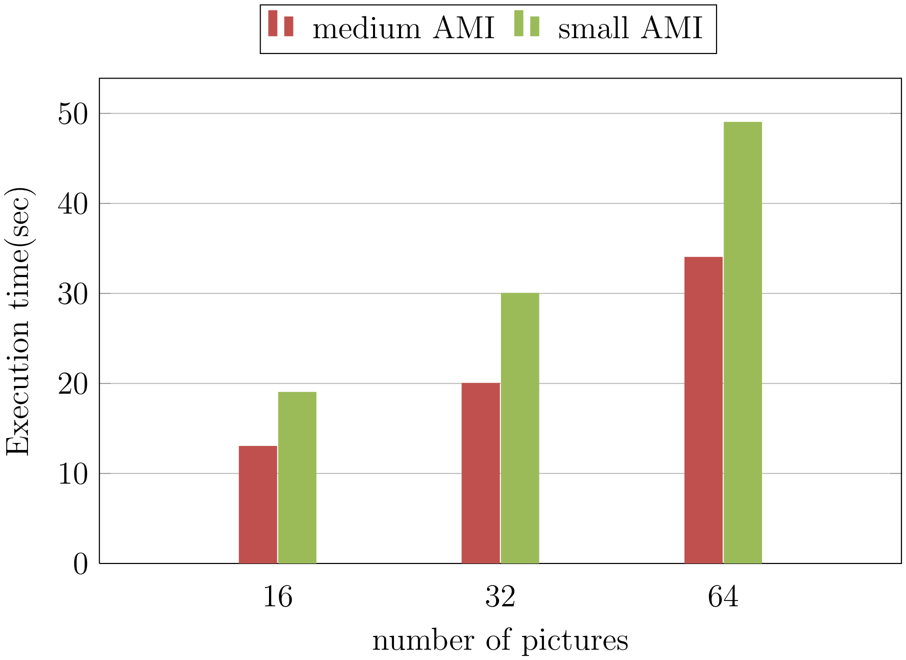


Fig. 6.2.: Execution time of face recognition application with local-only data for small vs. medium AWS machine instances.

6.3 Offloading Decision-Making

6.3.1 Offloading Manager

One of the most important features affecting the performance of an MCC framework is its ability to adapt to changing network conditions when making offloading decisions. Offloading decisions in AAMCC are made by the offloading (execution) manager mentioned in chapter 4. The offloading (execution) manager is a service running on the mobile device, which is responsible for dynamically determining whether a specific application module should be migrated to a cloud server to minimize execution time, given the execution tree of the application. An execution tree, the structure of which is seen in figure 6.4, is a representation of the interactions between the modules of an application. The node at the top of the tree is the application entry node (i.e. the main module). If a node is the direct child of another node in the execution tree, the module represented by the parent node makes a direct invocation to the

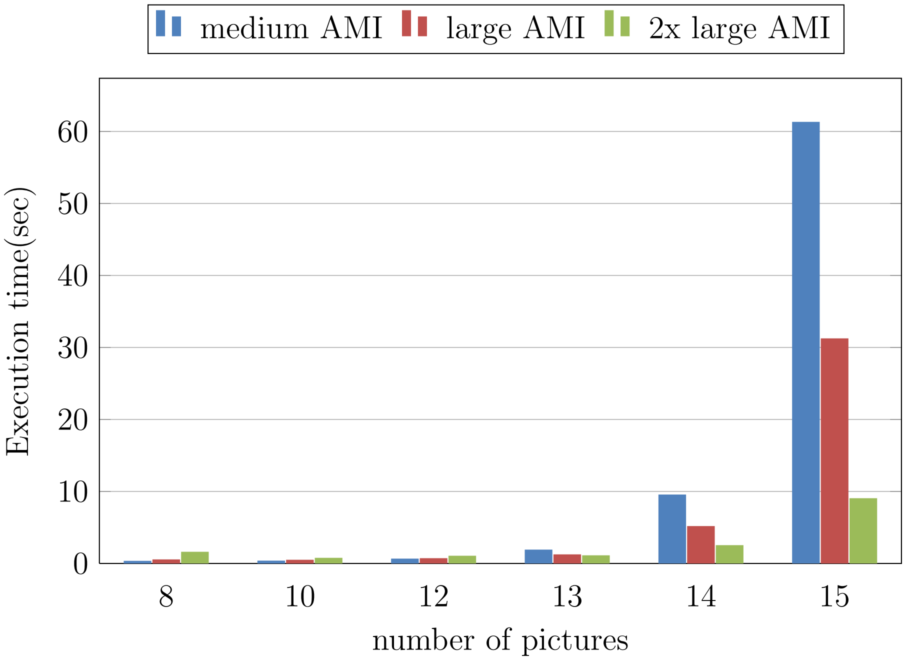


Fig. 6.3.: Offloaded execution time of multi-threaded NQueens solver for (a) medium, (b) large and (c) 2x large AWS machine instances.

module represented by the child node. Note that a tree-based representation of the interactions between application modules is based on the assumption that there are *no cyclic dependencies*, which is a good programming practice, between any modules. Also, any application module invoked by multiple different modules is represented as a different node for each dependency (i.e. replicated), to avoid increased complexity of the offloading optimization problem.

Each node x in the execution tree is annotated with a number m_x. m_x is the time to execute the module x on the mobile device less the time to execute its submodules. Each edge in the tree is annotated with a number c_x, which is the total size of the data that flows into module x from its parent module and back for the invocation of x. Note that although the structure of the execution tree would not change between different executions, the edge annotations are determined at application runtime, as the execution time and size of data to be transferred between modules could be functions of the problem size. Note that this representation does not include numbers

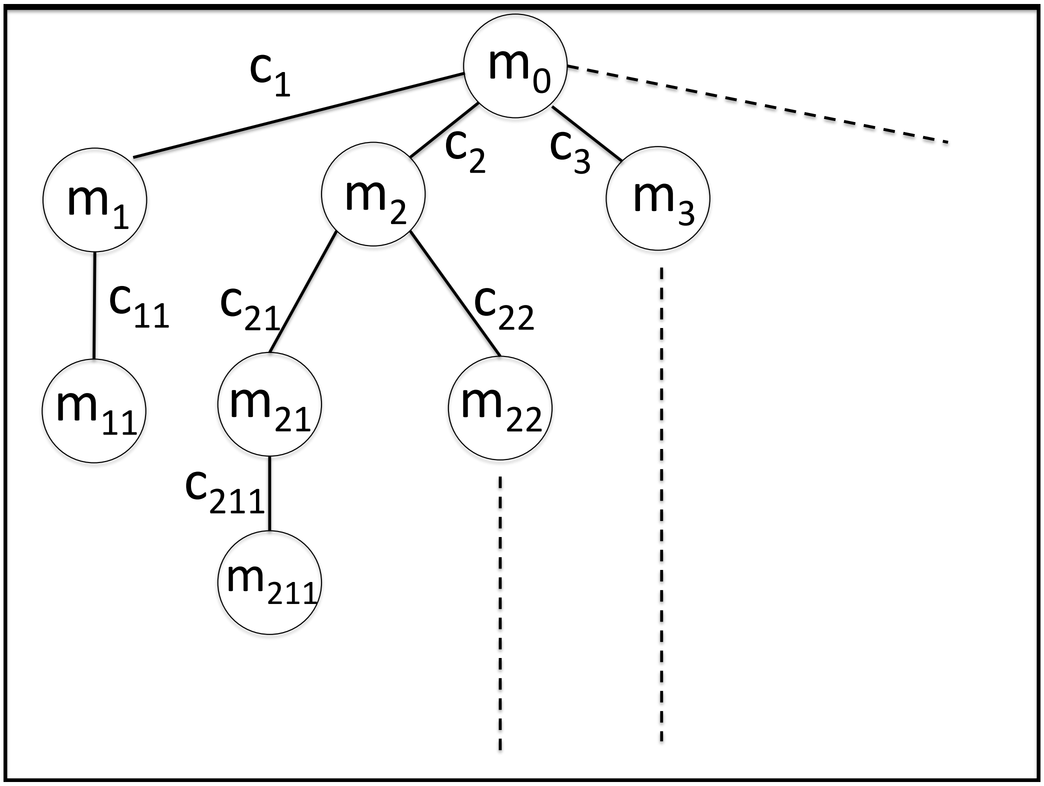


Fig. 6.4.: Execution tree structure.

for execution time of a module in the cloud. Because the execution plan only includes offloadable modules, execution time in the cloud is considered negligible compared to that on the device.

The offloading manager uses the following cost model to make offloading decisions for each offloadable application partition (Note that a submodule s of an application module x is a module invoked by x):

cd_x: the cost (in execution time) of executing application partition x locally on the device

cc_x: the cost (in execution time) of executing application partition x in the cloud

b: the network bandwidth available to the application

x_s: the set of submodules of x.

Then cd_x and cc_x are calculated as follows:

$$cd_x = m_x + \sum_{i \in x_s} argmin(cd_i, cc_i) \tag{6.1}$$

$$cc_x = c_x / b \tag{6.2}$$

In order to determine m_x, we currently use a static application profiler measuring the execution time of each application partition and record the results as metadata of the application, so they are available for use by the execution manager during application execution.

As seen in equation 6.1, the optimization problem has a recursive formulation: To determine the execution cost of an application module, we first need to determine the execution costs of its submodules. However, the offloading decisions regarding the submodules of a module could become invalid once the costs of executing the module on the device and in the cloud are calculated, as offloading the module implies offloading its submodules too. We solve the optimization problem with a depth-first traversal of the execution tree, updating the local and offloaded execution cost of each module once all of its submodule execution costs are set. Algorithm 2 shows how to solve the makespan optimization problem. Note that, once the whole execution tree is traversed using this algorithm, all children of a node chosen to be offloaded are offloaded with it as well (even if the first decision for them during the traversal was to keep them local).

Algorithm 2: Algorithm for making offloading decisions

```
input  : tree: Execution tree, where each node is annotated with the
         local execution cost m and offloading data transfer cost d
output: execution plan containing offloading decisions for each node in
         the tree
$parentNode \leftarrow tree.root$;
while parentNode has more children do
    $childNode \leftarrow parentNode.nextChild$;
    if $childNode.nextChild = null$ then
        if $childNode.m > childNode.d$ then
            $childNode.decision \leftarrow offload$;
            $childNode.cost \leftarrow childNode.d$;
        else
            $childNode.decision \leftarrow local$;
            $childNode.cost \leftarrow childNode.m$;
        end
        $parentNode.m \leftarrow parentNode.m+childNode.cost$;
        if $parentNode.nextChild = null$ then
            if $parentNode.m > parentNode.d$ then
                $parentNode.decision \leftarrow offload$;
                $parentNode.cost \leftarrow parentNode.d$;
            else
                $parentNode.decision \leftarrow local$;
                $parentNode.cost \leftarrow parentNode.m$;
            end
            $parentNode \leftarrow parentNode.parent$;
        end
    else
        $parentNode \leftarrow childNode$;
    end
end
```

6.3.2 Experiments with Offloading Manager

Experiments were performed with a synthetic application consisting of 6 offloadable modules to evaluate the performance of the offloading manager in making effective offloading decisions. Each module in the application has a different execution time on the mobile device and different amount of data transfer requirements for offloading. Figure 6.5 shows a comparison of the total execution times of the application for the cases of (a) monolithic execution on the device, (b) complete offloading to the cloud, and (c) using the offloading manager to decide which modules to offload to the cloud, where the device's available bandwidth is upper-bounded by the connection speed types on the x-axis. The values for the different connection types is provided in table 6.4. The experiments were run 5 times under stable conditions on a Motorola Atrix 4G device for each connection speed type, and we report the average results.

Table 6.4: Connection speeds used in the offloading manager experiments

Connection type	downlink bandwidth	uplink bandwidth	DNS delay	downlink delay	uplink delay
Wi-Fi	40 mbps	33 mbps	0 ms	1 ms	1 ms
Cable	6 mbps	1 mbps	3000 ms	2 ms	2 ms
3G	780 kbps	330 kbps	0 ms	100 ms	100 ms

As seen in figure 6.5, using the proposed offloading manager for execution platform decisions always achieves better (or same) performance than both monolithic execution on the device and offloading all modules to the cloud.

6.4 Makespan Estimation

The ability to make a close estimation of the *makespan* (the time taken to run a program segment) of an application module is important for an MCC model to be effective. While most of the previously proposed makespan estimation models for MCC are based on the assumption that the execution time of an application module offloaded to the cloud will be negligible, this assumption may not always hold due to

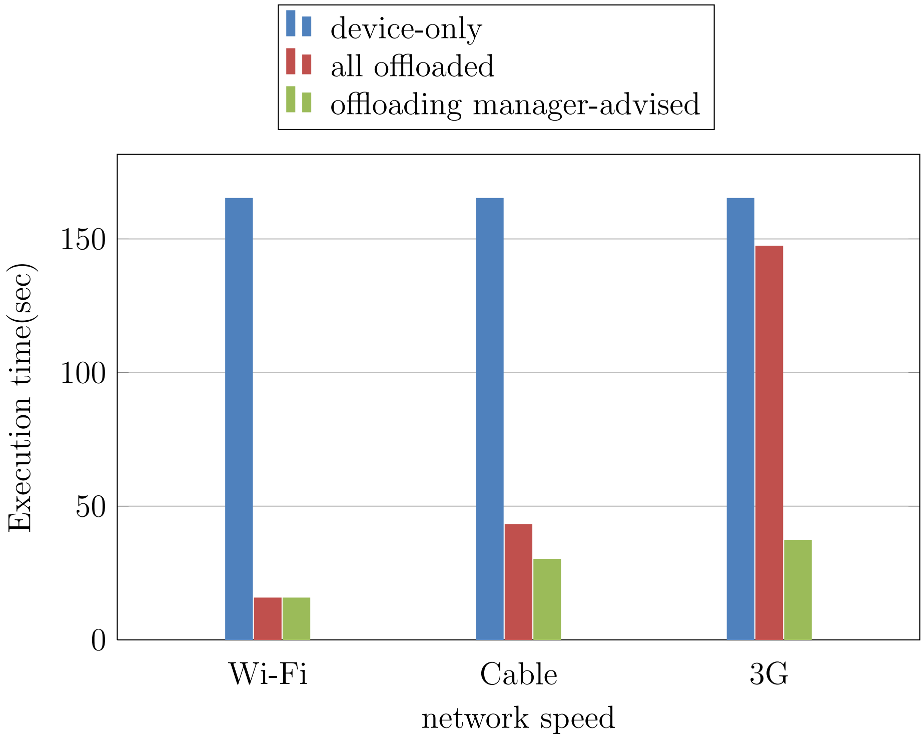


Fig. 6.5.: Comparison of the average execution times for a multi-module synthetic application in the case of (a) device-only execution, (b) complete offloading and (c) offloading manager-advised execution for different network speeds.

the dynamic nature of cloud platforms and the lack of information about the platform characteristics in some cases. In section 6.2.3, we showed that the characteristics of VMIs in the cloud can have significant effects on the performance of the applications they are hosting. Additionally, because cloud resources are shared between multiple users, the existence of other users on the same platform can significantly affect the makespan of an application in the cloud as well.

Estimation of the makespan of an application module serves the following purposes in the context of MCC:

- The estimated makespan can be compared to an expected value to see whether the execution is likely to satisfy specific time constraints. In the case of a low-performance platform, the computation can be migrated to a different platform.

- In the case of continuous or periodic makespan estimation updates, fluctuations in the estimates can provide an idea about the sharing of resources on the cloud platform.

6.4.1 Execution Profiling Model

Software execution profiling is an active area of research, as runtime performance of applications is of utmost importance for many systems. Although profiling the execution of an application under stable conditions on a specific architecture can yield accurate estimations for the makespan of the application under the exact same conditions, getting an accurate estimation under highly dynamic conditions and for platforms with different architectures is much more difficult. One of the main approaches used for makespan estimation is low-level analysis of a program to identify all instructions in the program and calculate the number of CPU cycles the program would take to execute, which then translates into a specific total execution time based on the architecture of the platform [68]. However, this approach is too specific and requires knowledge of the exact architecture of the target platform. It also does not account for the dynamicity of the runtime environment. In order to achieve accurate results, a makespan estimation model should continuously update its estimates based on changing runtime conditions, and not rely on a specific architecture. Fitness for MCC also requires that the model not be too complex, so that real-time performance requirements can be satisfied.

The model we propose for makespan estimation of an application module is a simple, platform-independent model based on heuristics affecting makespan. The main idea is that the relative makespans of different execution blocks in an application should be fixed under stable runtime conditions. This means that, if it takes x seconds to execute a code block $b1$ on machine $m1$, and $2x$ seconds to execute block $b2$ on the same machine, then $b2$ will take $2y$ seconds to execute on another machine where $b1$ takes y seconds. There are a few exceptions to this:

- Code blocks involving network activity, since their makespan is highly dependent on the network conditions.
- Code blocks involving conditionals with branches of significantly different computational complexity.
- Code blocks accepting input parameters of significantly different sizes at runtime (at different runs), where the input size determines computational complexity.

The makespan estimation model is trained and used in the application code as follows:

1. The list of exceptions above forms the list of heuristics we use in the model, i.e. when selecting the code blocks whose makespans are to be integrated into the estimation model, we eliminate such blocks from the candidate block list.
2. For the remaining code blocks, we run the application multiple times on the same platform and record the average makespan for each block.
3. Performance checkpoints corresponding to the selected code blocks are inserted into the application.
4. During application execution, each performance checkpoint records the actual makespan of the corresponding code block at runtime. It also makes an estimation of the makespan of the next code block with a performance checkpoint, based on the relative makespans of the two code blocks.

Given an application with n performance checkpoints, the makespan estimation for code block number x is calculated using equation 6.3:

$$T_{est}(x) = T_{true}(x-1) \times \frac{T_{avg}(x)}{T_{avg}(x-1)} \tag{6.3}$$

In equation 6.3:

T_{est}: Makespan for a code block estimated using the model

T_{avg}: The average makespan for a code block obtained from the statistics

T_{true} : The actual makespan for a code block recorded at runtime

The estimated total makespan for the remaining code blocks after x is calculated using equation 6.4:

$$T_{remaining}(x) = \sum_{i=x+1}^{n} T_{true}(x) \times \frac{T_{avg}(i)}{T_{avg}(x)} \tag{6.4}$$

6.4.2 Self-Performance Evaluating Agents

In order to be used in AAMCC, the proposed makespan estimation model needs to be integrated into the autonomous application modules, so that they can monitor their own performance during execution on a cloud platform and take appropriate actions based on the estimated remaining execution time. The integration is performed in a manner similar to the integration of integrity checkpoints into agent code, described in section 5.6, using AOP. In the case of self-performance evaluating agents, during agent execution an updated estimate of the remaining makespan is calculated at each performance checkpoint. The agent can then use this information to clone itself to a different platform if the performance is significantly below an expected threshold. Note that the agent code is transformed into a stateful program as described in section 5.6.3 to use this cloning approach, so that the time spent on the already executed code blocks is not wasted. States correspond to the different performance checkpoints.

6.4.3 Experiments with Execution Profiler

We evaluated the performance of the makespan estimation model with the face recognition application running on different VMIs in EC2 (VMI specifications listed in table 6.5). 9 performance checkpoints were inserted into the application, which correspond to the major method calls. Each experiment was repeated 100 times and we report the average numbers over those runs. The face recognition application was the only process running on the VMIs in this set of experiments.

Table 6.5: Amazon EC2 instance type specifications for makespan estimation experiments [67]

Instance type	memory	number of cores	network performance
m1.small	1.7 GB	1	low
m1.medium	3.75 GB	1	moderate
m1.large	7.5 GB	2	moderate
c1.medium	1.7 GB	2	moderate

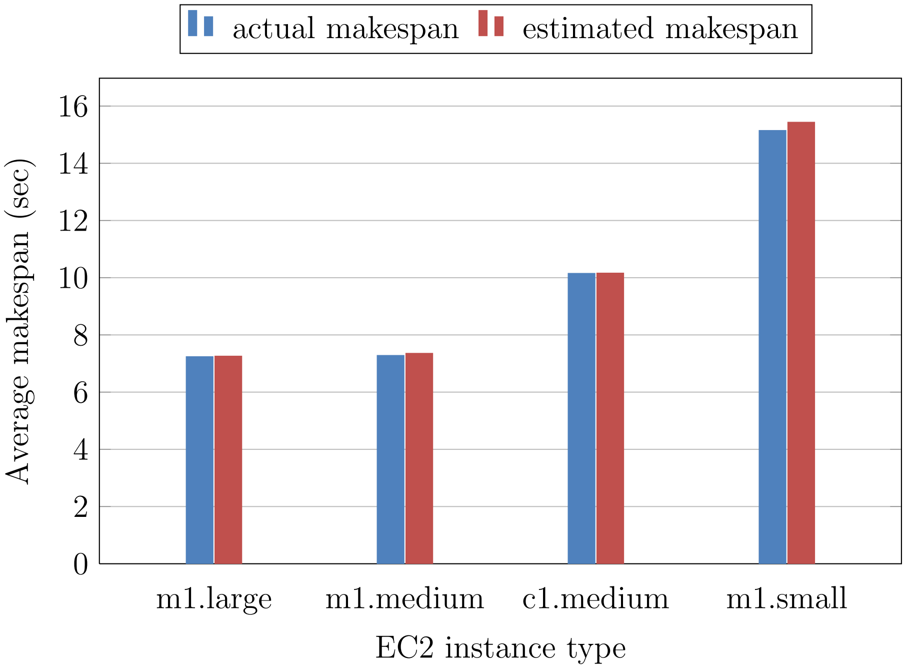


Fig. 6.6.: Comparison of average actual and estimated total makespans for the face recognition application for different VMI types in EC2.

Figure 6.6 provides a comparison of the total actual and estimated makespans of the face recognition application for different types of VMIs in Amazon EC2. As seen in the figure, the actual average makespans and the average makespans estimated by the proposed model are very close to each other for all VMI types. For each VMI type, we also calculated the absolute value of the difference between the true makespan and the estimated makespan at each data point. Table 6.6 summarizes the average of the absolute errors and the ratio of the average absolute error to the average true makespan for each VMI type. As seen in the table, the average absolute error is less than or equal to 292 ms, which translates into less than 2% of the total makespan for all VMI types, which is negligible. We also observe that the error increases slightly on VMIs with smaller computing power. These results prove that the proposed makespan estimation model is an effective tool for MCC, under stable conditions in the cloud.

Table 6.6: Makespan estimation errors for different VMI types

Instance type	avg. true makespan (ms)	avg. absolute error (ms)	absolute error percentage (%)
m1.small	15142	292	1.9
c1.medium	10147	125	1.2
m1.medium	7277	132	1.8
m1.large	7234	75	1.0

We also performed experiments to see the effect of integrating the proposed self-performance evaluation approach into autonomous application modules in AAMCC, as described in section 6.4.2. Experiments were performed with the face recognition module always offloaded to a micro instance in EC2. Based on the performance evaluation on the micro instance, the offloaded agent made the decision regarding whether to clone itself to another VMI in EC2. For the experiments, the only other process running on each VMI was a CPU-intensive C process (the same process on all instances).

Table 6.7: Average execution time savings (in seconds) of self-performance evaluation for face recognition module cloned from a micro instance

m1.large	c1.medium	m1.medium	m1.small
78.70	74.90	71.94	62.04

Table 6.7 shows the average execution time savings obtained by using the self-performance evaluation and cloning approach, for different types of machine instances as the second hop (the VMI where the agent is cloned to). The average total makespan without dynamic performance monitoring in the experiments was around 101 seconds. As seen in the table, the proposed model is able to achieve between 61% - 73% time savings by making the decision to clone an agent under unfavorable runtime conditions.

6.5 Chapter Summary

In this chapter we discussed the importance of context-aware decision-making on the performance of MCC frameworks and proposed a bandwidth-aware offloading approach for multi-component mobile applications in AAMCC, as well as a model for performance-aware execution of autonomous agents in the cloud. The proposed offloading management approach was shown to provide better performance than both on-device execution and complete offloading under various network conditions. We also showed that the proposed dynamic makespan estimation model can achieve high performance under stable conditions in the cloud, which allows autonomous application modules to monitor their own performance and make effective migration decisions to minimize application makespan.

7. APPLICATION: MOBILE-CLOUD NAVIGATION GUIDE FOR THE BLIND

7.1 Motivation

There were about 21.2 million visually-impaired people in the United States in 2011, according to the survey by the National Center for Health Statistics [69]. The 2009 survey by the U.S. Bureau of Labor Statistics [70] states that at all levels of education, persons with a disability were less than half as likely to be employed than were their counterparts with no disability and 75% of the blind population in the survey were not part the labor force. The two biggest challenges for independent living of the visually-impaired as stated in [71] are access to printed material and safe and efficient navigation. In order to navigate safely, blind people must learn how to detect obstructions, find curbs when outside and stairs when inside buildings, interpret traffic patterns, find bus stops and know their own location [71], i.e. be fully aware of the context of their environment. Fully context-aware navigation for the blind is not only safely reaching a destination, but also being able to track personal items/objects of use (such as luggage on an airport carousel), and identify/interact with acquaintances on the way or at a social gathering.

Independent navigation is becoming a more challenging task for blind people every day with the advances in technology, products of which such as hybrid cars (aka quiet cars), makes it more difficult to rely on other senses such as hearing for safety [72]. Hence, while technology is thought to improve the lives of many people in the world, it often causes people with disabilities to fall behind, sometimes even putting their lives at risk. Most navigation aids utilizing high technology have high price tags (a few thousands of dollars), making them inaccessible to the majority of the blind population. The difficulty of independent navigation in the increasingly complex urban world and the lack of affordable navigation technology cause isolation of the visually-impaired. Blind people, not being able to drive vehicles, are the ones in greatest need for accessible transportation services. However, as stated by Kwan et al. [73], in many cities, people with disabilities are seldom seen using the street or public transportation due to insufficient accessibility. This situation has not changed significantly despite requirements of compliance with special accessibility rules in buildings and facilities vehicles (e.g. announcing major stops in public transportation vehicles) such as those stated in the Americans with Disabilities Act Accessibility Guidelines [74].

As reported by the World Health Organization, more than 82% of the visually-impaired population in the world is age 50 or older [75], and the old population forms a group with a diverse range of abilities [76]. This diversity makes it difficult to develop a single navigation device usable by the whole blind population. Devices requiring an involved training process are especially not appropriate for the elderly people with relatively limited learning capability compared to the young. The design of a self-complete navigation aid for the visually-impaired thus requires consideration of many aspects of the problem including wide usability –in terms of both ease of use by a group of diverse learning abilities and minimal infrastructural requirements–, real-time processing capability, providing maximal context-awareness, minimal obtrusiveness, optimal interface, as well as preservation of the privacy of the users and the people around them.

Cloud computing is very promising to make computerized systems increasingly capable, as it provides access to huge amounts of information and computation resources without having to own them. Cloud computing has significant implications for the advancement of truly accessible and effective technology for context-aware navigation of the visually-impaired, as it obviates the need to process information locally with limited resources and makes it possible to provide various functionalities on a single handheld device. Use of cloud computing also makes it possible to integrate data from various resources for more accurate guidance as opposed to approaches relying on a single source of data (such as pure image processing). Aware of the fact that the interplay between mobile and cloud computing has the potential to solve many problems remaining largely unsolved for the blind and visually impaired community, we focus on applications that take a mobile-cloud computing approach to context-aware navigation in this chapter.

7.2 Proposed Context-Aware Navigation Architecture

The basic mobile-cloud system architecture we propose for building the context-aware navigation functionality consists of the main components seen in Figure 7.1. Visual context data is captured by camera modules integrated into eyeglasses and fed to the mobile device through an appropriate interface (recent technologies such as Google Glass [77] combining these functionalities in a single device facilitate this interaction). A speech interface on the handheld device takes commands from the user and context-awareness guidance is achieved with local computation on the mobile device with an integrated compass and positioning module as well as computation in the cloud. Context-relevant feedback is provided to the user via an auditory interface on the mobile device. Machine instances in the cloud run different context-awareness tasks, each acting as an integrator of context relevant data from the mobile device and various resources on the web to provide accurate guidance.

7.3 Pedestrian Crossing Guide

The ability to detect the status of pedestrian signals accurately is an important aspect of providing safe guidance during navigation. The inherent difficulty of the problem is the fast image processing required for locating and detecting the status of the signals in the immediate environment. As real-time image processing is demanding in terms of computational resources, mobile devices with limited resources fall short in achieving accurate and timely detection. An accurate pedestrian signal status detection service would benefit not only the visually-impaired, but also

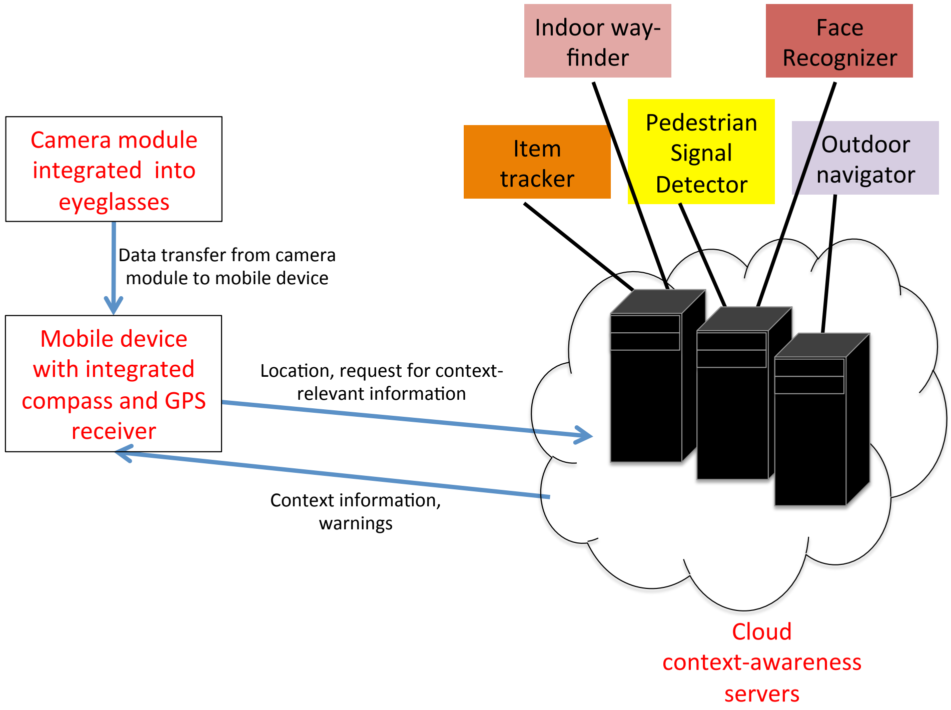

Indoor way-finder
Face Recognizer
Camera module integrated into eyeglasses
Item tracker
Pedestrian Signal Detector
Outdoor navigator
Data transfer from camera module to mobile device
Mobile device with integrated compass and GPS receiver
Location, request for context-relevant information
Context information, warnings
Cloud context-awareness servers

Fig. 7.1.: Proposed mobile-cloud architecture for context-aware blind navigation

the color-blind as well as systems like autonomous ground vehicles and even careless drivers. This problem has been studied by many researchers including [78–82]. The major shortcoming of some of the previously proposed approaches is their reliance on a low-portability computation device for the necessary image processing, and yet others are lacking the universal design principle by limiting their training data to a specific set and draining the battery of the mobile device by running the detection algorithm on the device. On the other hand, although the approach proposed by Bohonos et al. [83], not relying on image processing, is promising for accurate detection, it requires installation of special hardware at pedestrian signals, which limits its use to a very small area.

Our initial efforts for the proposed context-aware navigation system focused on the development of a pedestrian signal detector with an Android-based mobile phone and a crossing guidance algorithm running on a machine instance in the Amazon

EC2. We developed the system by integrating crossing guidance into an outdoor navigation application, WalkyTalky, released by the Eyes-Free group at Google [84]. The pedestrian signal detector of the developed system uses a cascade of boosted classifiers based on the AdaBoost algorithm [85], which is popular for real-time object recognition tasks, and haar-like features [86] to detect the presence and status of pedestrian signals in a picture captured by the camera of the Android mobile phone. The flow of events in the system is as follows:

1. The Android application relies on the GPS receiver on the phone and the Google Maps [87] server to detect the current location of the blind user.

2. Once the application detects that the blind user is at an urban intersection, it will trigger the native camera to take a picture (or multiple consecutive pictures) and send the picture to the server running on the Amazon EC2 cloud.

3. The server performs the image processing, applying the *pedestrian signal detection* algorithm and returns the result as to whether it is safe for the user to cross.

An important aspect of the pedestrian signal status detection problem are timeliness of response and accuracy. The real-time nature of the problem necessitates response times of less than 1 second, while high accuracy of detection should be achieved to ensure safety of the user. Experiments were performed to test the response time of the pedestrian signal detector application developed. Test data used in the experiments consists of pictures at outdoor locations at the Purdue University campus, which include scenes of different pedestrian signals. The application developed was installed on an Android mobile phone, connected to the Internet through a 4G network. The sample task in the experiments involved processing five different resolution level versions of pictures. The average response times that were determined by the time period between capturing a frame and receiving the response from the server running at Amazon Elastic Compute Cloud about the pedestrian signal status, were measured for each frame resolution level as determined by a Java platform-specific measure. A resolution level of 0.75 stands for the original frame as captured by the camera, whereas the lower resolution levels represent compressed versions of the same set of frames, where image quality falls with decreasing resolution levels. The response times for different resolution levels are seen in figure 7.2. Response times for the original frames are around 660 milliseconds on average, which are acceptable levels for the real-time requirements of the problem. We also see that response time decreases further when lower-quality, compressed versions of the frames are sent to the remote server instead of the originals.

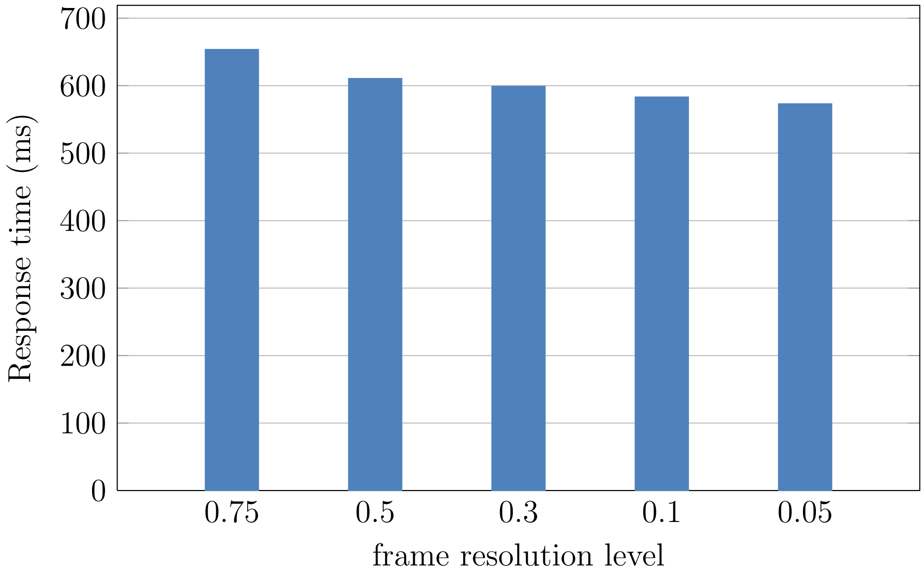


Fig. 7.2.: Response times of the pedestrian crossing guide application for different frame resolution levels.

7.4 Other Possible Applications of MCC to Assistive Technologies for the Blind

7.4.1 Indoor Way-finding

Indoor navigation in unfamiliar environments is a difficult task even for people without visual impairments. Blindness makes this task harder with the added complexity of the presence of obstacles, stairs, doorways, etc. Accurate positioning is one of the main challenges faced by indoor navigation aids, while another major challenge is the absence of maps such as those available for streets. The absence of GPS signals indoors necessitates the use of some other positioning technology, the common one being Wi-Fi tracking. However, Wi-Fi has been shown to provide accuracy at most at the room level [88], which is not sufficient for more refined tasks such as going to a specific seat in a specific room.

An MCC approach for indoor navigation has the advantage of being able to combine information from many resources to guide the user to find his/her way in an unfamiliar indoor environment. Whenever maps of the location in question are available online (The Micello [89] company is already mapping important indoor locations worldwide), they can be used for guidance and in other cases image processing will help to detect people in the surroundings to ask for help or optical character recognition will help to detect signs in the building leading to various places. Use of cloud computing also makes it possible to form a large online database of places visited by different users, such that after the first visit to that place, different features of the place can be logged in the database for use at subsequent visits to the same place.

7.4.2 Recognition of Objects in Complex Scenes

Real-time and accurate classification of objects in highly complex scenes is an important problem, having been the recent focus of attention of the computer vision community due to its many application areas. A solution to this problem is particularly significant for the visually-impaired, as they have great difficulty reaching desired objects especially in unfamiliar environments. While boosting methods with the sliding window approach provide fast processing and accurate results for particular object categories, they cannot achieve the desired performance for other more involved categories of objects. Recent research in computer vision has shown that exploiting object context through relational dependencies between object categories leads to improved accuracy in object recognition. Integration of the pair-wise co-occurrence frequencies of object categories has proven effective to classify objects when the categories of their neighbors in the scene are known. While efforts in collective classification of objects in images have resulted in complex algorithms, the real-time nature of the problem requires the use of simpler algorithms with accurate results. Use of mobile-cloud computing is promising for a real-time solution to this problem, as the user will no longer be limited to the computational power of a mobile device for object classification in highly complex scenes and will be able to utilize the high processing power of the cloud resources to obtain accurate results.

7.4.3 Recognition of People Augmented with Social Networks and Location

Blind people have limited opportunities for social interaction compared to those without visual impairments. Aiding social interaction for blind people has not received as much attention by assistive technology researchers as other topics, such as indoor/outdoor navigation, although it is a major component of context-awareness, except for a few proposals including a system using a haptic belt to convey non-verbal communication cues during social interactions to individuals who are blind [90]. Identifying familiar people in the immediate surroundings is a significant aspect of context-awareness and a system capable of this task is highly desirable by the blind, as it is expected to help with social interaction. Recognizing gestures makes communication even more effective.

Research in face detection mainly built on the state-of-the-art Viola-Jones algorithm [91], has proven quite successful and is utilized by many real-world systems today, such as digital cameras and online social networks. The release of depth cameras is promising for further advances in the field of face recognition, as these cameras provide a three-dimensional pixel map of the face and make it easier to detect certain facial features with the extra information of pixel depth.

We believe robust person recognition can only be achieved with a combination of clues indicating the presence of a specific person in the immediate environment instead of solely relying on face recognition. One of those clues is the current location of a person as determined by GPS and another one is the information that can be retrieved from online social networks that the person is a member of. This multi-cue approach to person recognition can be made possible by using the combined resources of mobile and cloud computing.

We propose a mobile-cloud based system for person recognition with a face recognition algorithm running in the cloud and where an Android phone is used to capture an image of the environment and send it to the cloud server for matching against a dataset of friends' pictures for a specific person. Higher performance is expected upon extension of the system to include location information retrieved from the profiles in the user's social network to figure out patterns such as the frequent co-occurrence of specific people at the same location or information as to where a specific person currently is.

7.5 Chapter Summary

In this chapter, we proposed an open, extensible mobile-cloud computing architecture to enable context-aware navigation for the visually-impaired. The proposed system is based on a collaboration model between everyday mobile devices, the wealth of location-specific information resources on the web, and the computational resources made available by major cloud computing providers. The proposed architecture allows for richer context-awareness and high quality navigation guidance. We discussed major possible functionalities of the proposed system and showed promising results for real-time responses from the proposed architecture in the context of a pedestrian crossing guide.

8. SUMMARY

8.1 Contributions

This dissertation focuses on the problem of effective, efficient and secure computation offloading and execution in mobile-cloud computing.

Our main contributions with this dissertation can be summarized as follows:

1. **Efficient computation offloading framework for mobile-cloud computing:**

   We proposed a dynamic code offloading framework for mobile-cloud computing, based on autonomous agents. Our approach does not impose any requirements on the cloud platform other than providing isolated execution containers, and it alleviates the management burden of offloaded code by the mobile platform using autonomous agent-based application partitions. The experiments we performed with real-world applications using the Amazon EC2 cloud show that the framework achieves significantly higher performance with regards to execution time compared to monolithic application execution on a mobile device, in the computationally intensive cases of the applications.

2. **Low-overhead tamper-resistance approach for mobile-cloud computing:**

   We proposed a dynamic tamper-resistance approach for protecting mobile computation offloaded to the cloud, by augmenting mobile agents with self-protection capability. The proposed approach uses integrity verification units integrated transparently into application code, which are used for program introspection during runtime. The framework uses a special mechanism for the encryption of messages to be sent from the cloud to the mobile platforms, which allows for the detection of tampering with messages in transit or with the integrity verification units during their execution in the cloud. We showed through experiments that the tamper-resistance framework achieves very low execution time overhead and is capable of detecting both load-time and runtime modifications to agent code, as well as taking the appropriate actions upon tamper detection.

3. **Context-aware computation migration and execution approach for mobile-cloud computing:**

   We investigated the effects of different runtime environment conditions (context) including varying network bandwidth, multi-tasking and different machine architectures, on the makespan performance of offloaded computation in mobile-cloud computing. We proposed a simple and low-overhead dynamic makespan estimation model for computation offloaded to the cloud, that can be integrated into mobile agents to enhance them with self-performance evaluation capability. Experiments we performed using a real-world application with different virtual machine instance types in the Amazon EC2 cloud show that the proposed model is capable of estimating makespan with high accuracy.

4. **Application of mobile-cloud computing to assistive technologies:**

   We proposed novel applications of mobile-cloud computing for helping context-aware navigation by visually-impaired people. Specifically, we performed a feasibility study for using real-time mobile-cloud computing for the task of guiding blind users at pedestrian crossings with no accessible pedestrian signal. The system we propose for this task is based on the recognition of the status of pedestrian signals at crossings, using an object recognition algorithm running in the cloud, communicating with the mobile device of the user. We presented initial performance results of the developed navigation application, which proves that mobile-cloud computing is promising for assistive technologies with real-time requirements.

The mobile-cloud computation framework we introduced in this dissertation is an important tool for high-performance, secure mobile-cloud computing, providing the chance for widespread adoption due to its ease of integration into many different mobile and cloud platforms. The results of the work in this dissertation are also highly relevant for other fields of computer science. The major broader impacts of this work can be summarized as follows:

- The proposed computation offloading model offers a new way of thinking about mobile application programming, which could be extended in various directions to achieve real-time, high-performance mobile computing.
- The principles of the proposed tamper-resistance approach apply not only to MCC, but also to secure (tamper-resistant) cloud computing in general. The techniques introduced in this work are promising to provide security solutions for program execution in untrusted cloud environments, which will allow for higher confidence in the adoption of cloud computing for various computing needs.
- The proposed self-performance evaluation approach for autonomous agents is highly relevant for high-performance agent-based distributed computing. Its simplicity and reliance on standard technologies makes it an effective tool for various distributed computing problems.
- Experiments and observations of this work about the importance of context provide important guidance for future work in the field of real-time mobile-cloud computing.

8.2 Future Work

The mobile-cloud computation framework we introduced in this dissertation can be extended in a number of ways to address other challenges peculiar to MCC. Below is a list of tasks we plan to work on to integrate solutions into the proposed framework for some of those major challenges.

1. **Offloading manager with energy constraints**

   The current framework makes offloading decisions for applications components in a way to minimize the total makespan of the application. While minimizing the execution time is an important factor in real-time mobile computing, an equally important factor is the energy consumption on the mobile device. As the experiments in chapter 4 suggest, data transfer to/from the cloud can consume a significant amount of energy. In future work, we plan to extend the optimization model of the execution manager to include energy constraints. This will be achieved by transforming the optimization problem into a constraint satisfaction problem, where the constraints will ensure that the total energy consumption with offloading does not exceed the total energy for on-device execution.

2. **Makespan estimation of on-device execution**

   The current framework uses static values for the on-device makespan of application components as determined by previous runs of the application on the mobile device. While this approach provides an accurate estimation of the on-device makespan under stable conditions on the exact same mobile device, makespan values could vary significantly across devices with different characteristics and different runtime conditions. In future work, we plan to integrate a dynamic makespan estimation model, much like the one we introduced in chapter 6, into the execution manager component of the framework to estimate the on-device makespan of application components. This approach will allow for generalization of makespan estimation across a range of devices, and obviate the need to gather statistics for different devices before application installation.

3. **Generalization of cost estimation model for different problem sizes**

   In the current framework, the execution cost of application components (in terms of time) is measured statically as mentioned in the above item. Although this approach works well with problems of a pre-determined size, it does not allow for generalization to different sizes of the same problem, i.e. it does not integrate the computational complexity of a problem to estimate the execution time for different instances (input sizes) of that problem. In future work, we plan to integrate a computational complexity component into the execution manager to estimate the makespans of application components for dynamic problem input sizes. This approach will provide yet another level of generalization, and a much needed one for dynamic MCC.

4. **Automatic selection of best tamper-resistance integrity checkpoints**

   In our current tamper-resistance model, integrity checkpoints are placed at every method call in an application component offloaded to the cloud for execution. For applications densely populated with method calls, sparser placement of integrity checkpoints could achieve nearly as good tamper-resistance. In future work, we plan to design an algorithm for automated placement of integrity checkpoints in offloadable application components in a way that minimizes the number of integrity checkpoints, but provides the same level of security against tampering attacks.

5. **Automatic selection of best performance checkpoints for makespan estimation**

   In our current makespan estimation model, performance checkpoints are placed at every execution block that passes the inclusion test. Depending on the application structure, it might be possible to eliminate some of these checkpoints, such as very small blocks of code, which are not very suggestive for the remaining makespan of the application. It is also possible to group different blocks of code with similar characteristics (e.g. same operations) together to make even more accurate makespan estimations. In future work, we plan to design an algorithm to automatically select the best performance checkpoints for offloadable application components to maximize the accuracy of the makespan estimation under varying runtime conditions.

LIST OF REFERENCES

LIST OF REFERENCES

[1] H. Chu, H. Song, C. Wong, S. Kurakake, and M. Katagiri, “Roam, a seamless application framework,” *Journal of Systems and Software – Special Issue on Ubiquitous Computing*, vol. 69, pp. 209–226, January 2004.

[2] X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, and D. Milojicic, “Adaptive offloading inference for delivering applications in pervasive computing environments,” in *Proceedings of the IEEE International Conference on Pervasive Computing and Communications (PerCom'03)*, pp. 107–114, 2003.

[3] S. Gouveris, S. Sivavakeesar, G. Pavlou, and A. Malatras, “Programmable middleware for the dynamic deployment of services and protocols in ad hoc networks,” in *Proceedings of the 9th IFIP/IEEE International Symposium on Integrated Network Management*, pp. 3–16, 2005.

[4] G. Chen, B. Kang, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, and R. Chandramouli, “Studying energy trade offs in offloading computation/compilation in Java-enabled mobile devices,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 9, pp. 795–809, 2004.

[5] C. Wang and Z. Li, “A computation offloading scheme on handheld devices,” *Journal of Parallel and Distributed Computing*, vol. 64, no. 6, pp. 740–746, 2004.

[6] G. C. Hunt and M. L. Scott, “The Coign automatic distributed partitioning system,” in *Proceedings of the 3rd Symposium on Operating System Design and Implementation (OSDI'99)*, pp. 187–200, 1999.

[7] M. Stoer and F. Wagner, “A simple min-cut algorithm,” *Journal of the ACM (JACM)*, vol. 44, no. 4, pp. 585–591, 1997.

[8] S. Ou, K. Yang, A. Liotta, and L. Hu, “Performance analysis of offloading systems in mobile wireless environments,” in *Proceedings of the IEEE International Conference on Communications (ICC'07)*, pp. 1821–1826, 2007.

[9] M. Tang and J. Cao, “A dynamic mechanism for handling mobile computing environmental changes,” in *Proceedings of the 1st ACM International Conference on Scalable Information Systems*, 2006.

[10] B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, “Clonecloud: Elastic execution between mobile device and cloud,” in *Proceedings of the 6th ACM European Conference on Computer Systems (EuroSys'11)*, pp. 301–314, 2011.

[11] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, “Maui: Making smartphones last longer with code offload,” in *Proceedings of the 8th ACM International Conference on Mobile Systems, Applications, and Services (MobiSys'10)*, pp. 49–62, 2010.

[12] L. Yang, J. Cao, S. Tang, T. Li, and A. T. Chan, "A framework for partitioning and execution of data stream applications in mobile cloud computing," in *Proceedings of the 5th IEEE International Conference on Cloud Computing (CLOUD'12)*, pp. 794–802, 2012.

[13] D. Huang, P. Wang, and D. Niyato, "A dynamic offloading algorithm for mobile computing," *IEEE Transactions on Wireless Communications*, vol. 11, pp. 1991–1995, June 2012.

[14] S. Park, Y. Choi, Q. Chen, and H. Y. Yeom, "SOME: Selective offloading for a mobile computing environment," in *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER'12)*, pp. 588–591, 2012.

[15] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Proceedings of the 31st IEEE International Conference on Computer Communications (INFOCOM'12)*, pp. 945–953, 2012.

[16] C. Xian, Y. H. Lu, and Z. Li, "Adaptive computation offloading for energy conservation on battery-powered systems," in *Proceedings of the IEEE International Conference on Parallel and Distributed Systems*, vol. 2, pp. 1–8, 2007.

[17] X. Li, H. Zhang, and Y. Zhang, "Deploying mobile computation in cloud service," in *Proceedings of the 1st International Conference on Cloud Computing (CloudCom'09)*, pp. 301–311, 2009.

[18] F. Lombardi and R. Di Pietro, "Secure virtualization for cloud computing," *Journal of Network and Computer Applications*, vol. 34, no. 4, pp. 1113–1122, 2011.

[19] J. Szefer, E. Keller, R. B. Lee, and J. Rexford, "Eliminating the hypervisor attack surface for a more secure cloud," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, pp. 401–412, 2011.

[20] F. Zhang, J. Chen, H. Chen, and B. Zang, "Cloudvisor: Retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization," in *Proceedings of the 23rd ACM Symposium on Operating Systems Principles*, pp. 203–216, 2011.

[21] C. Li, A. Raghunathan, and N. K. Jha, "Secure virtual machine execution under an untrusted management OS," in *Proceedings of the 3rd IEEE International Conference on Cloud Computing (CLOUD'10)*, pp. 172–179, 2010.

[22] M. Brenner, J. Wiebelitz, G. von Voigt, and M. Smith, "Secret program execution in the cloud applying homomorphic encryption," in *Proceedings of the 5th IEEE International Conference on Digital Ecosystems and Technologies (DEST'11)*, pp. 114–119, 2011.

[23] T. Sander and C. F. Tschudin, "Protecting mobile agents against malicious hosts," in *Mobile Agents and Security*, pp. 44–60, Springer, 1998.

[24] K. Heffner and C. Collberg, "The obfuscation executive," in *Information Security*, pp. 428–440, Springer, 2004.

[25] D. Huang, X. Zhang, M. Kang, and J. Luo, "Mobicloud: Building secure cloud framework for mobile computing and communication," in *Proceedings of the IEEE International Symposium on Service Oriented System Engineering*, pp. 27–34, 2010.

[26] X. Zhang, J. Schiffman, S. Gibbs, A. Kunjithapatham, and S. Jeong, "Securing elastic applications on mobile devices for cloud computing," in *Proceedings of the ACM Workshop on Cloud Computing Security*, pp. 127–134, 2009.

[27] W. Itani, A. Kayssi, and A. Chehab, "Energy-efficient incremental integrity for securing storage in mobile cloud computing," in *Proceedings of the IEEE International Conference on Energy Aware Computing (ICEAC)*, pp. 1–2, 2010.

[28] W. Jia, H. Zhu, Z. Cao, L. Wei, and X. Lin, "SDSM: A secure data service mechanism in mobile cloud computing," in *Proceedings of the IEEE Conference on Computer Communications Workshops*, pp. 1060–1065, IEEE, 2011.

[29] S. Hsueh, J. Lin, and M. Lin, "Secure cloud storage for conventional data archive of smart phones," in *Proceedings of the 15th IEEE International Symposium on Consumer Electronics (ISCE'11)*, pp. 156–161, 2011.

[30] J. Yang, H. Wang, J. Wang, C. Tan, and D. Yu, "Provable data possession of resource-constrained mobile devices in cloud computing," *Journal of Networks*, vol. 6, no. 7, pp. 1033–1040, 2011.

[31] Z. Zhou and D. Huang, "Efficient and secure data storage operations for mobile cloud computing," in *Proceedings of the 8th IEEE International Conference on Network and Service Management (CNSM'12)*, pp. 37–45, 2012.

[32] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proceedings of the IEEE Symposium on Security and Privacy (SP'07)*, pp. 321–334, 2007.

[33] S. Xiao and W. Gong, "Mobility can help: Protect user identity with dynamic credential," in *Proceedings of the 11th IEEE International Conference on Mobile Data Management (MDM'10)*, pp. 378–380, 2010.

[34] R. Chow, M. Jakobsson, R. Masuoka, J. Molina, Y. Niu, E. Shi, and Z. Song, "Authentication in the clouds: A framework and its application to mobile users," in *Proceedings of the ACM Workshop on Cloud Computing Security*, pp. 1–6, 2010.

[35] I. Bilogrevic, M. Jadliwala, P. Kumar, S. S. Walia, J. P. Hubaux, I. Aad, and V. Niemi, "Meetings through the cloud: Privacy-preserving scheduling on mobile devices," *Journal of Systems and Software*, vol. 84, no. 11, pp. 1910–1927, 2011.

[36] P. Mell and T. Grance, "The NIST definition of cloud computing," *NIST Special Publication 800-145*, 2011.

[37] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: Architecture, applications, and approaches," *Wireless Communications and Mobile Computing*, vol. 13, no. 18, pp. 1587–1611, 2013.

[38] X. Luo, "From augmented reality to augmented computing: A look at cloud-mobile convergence," in *Proceedings of the IEEE International Symposium on Ubiquitous Virtual Reality (ISUVR'09)*, pp. 29–32, 2009.

[39] D. Chess, C. Harrison, and A. Kershenbaum, "Mobile agents: Are they a good idea?," in *Mobile Object Systems Towards the Programmable Internet* (J. Vitek and C. Tschudin, eds.), Springer, 1997.

[40] I. Ahmed and M. J. Sadiq, "Information subsystem congestion analysis of a wide area-networked manufacturing system using mobile agents," *Journal of Manufacturing Technology Management*, vol. 16, no. 7, pp. 753–764, 2005.

[41] Telecom Italia Lab, "Java agent development framework." `http://jade.tilab.com/`. Accessed: 2013-11-05.

[42] Telecom Italia Lab, "Wade." `http://jade.tilab.com/wade/html/home-getting.htm`. Accessed: 2013-11-05.

[43] Google Inc., "Android." `http://www.android.com`. Accessed: 2013-11-05.

[44] Amazon Web Services Inc., "Amazon elastic compute cloud." `http://aws.amazon.com/ec2`. Accessed: 2013-11-05.

[45] K. Darnok. `http://darnok.org/programming/face-recognition`. Accessed: 2013-11-05.

[46] Motorola Mobility LLC, "Motorola Atrix 4G." `http://www.motorola.com/us/consumers/Motorola-ATRIX-4G/72112,en_US,pd.html`. Accessed: 2013-11-05.

[47] M. Gordon, L. Zhang, B. Tiwana, R. Dick, Z. M. Mao, and L. Yang, "PowerTutor: A power monitor for Android-based mobile platforms." `http://ziyang.eecs.umich.edu/projects/powertutor/`. Accessed: 2013-11-04.

[48] C. Collberg and J. Nagra, *Surreptitious software: Obfuscation, watermarking, and tamperproofing for software protection*. Addison-Wesley Professional, 2009.

[49] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, pp. 199–212, 2009.

[50] H. Chang and M. J. Atallah, "Protecting software code by guards," in *Security and Privacy in Digital Rights Management*, pp. 160–175, Springer, 2002.

[51] L. ben Othmane and L. Lilien, "Protecting privacy of sensitive data dissemination using active bundles," in *Proceedings of the IEEE World Congress on Privacy, Security, Trust and the Management of e-Business*, pp. 202–213, 2009.

[52] J. Ametller, S. Robles, and J. A. Ortega-Ruiz, "Self-protected mobile agents," in *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 362–367, 2004.

[53] P. Angin, B. Bhargava, R. Ranchal, N. Singh, M. Linderman, L. ben Othmane, and L. Lilien, "An entity-centric approach for privacy and identity management in cloud computing," in *Proceedings of the 29th IEEE Symposium on Reliable Distributed Systems*, pp. 177–183, 2010.

[54] B. Bhargava, P. Angin, R. Ranchal, R. Sivakumar, A. Sinclair, and M. Linderman, "A trust-based approach for secure data dissemination in a mobile peer-to-peer network of AVs," *International Journal of Next-Generation Computing*, vol. 3, no. 1, 2012.

[55] N. Påhlsson, "Aspect-oriented programming." `http://oberon2005.oberoncore.ru/paper/np2002.pdf`. Accessed: 2013-11-05.

[56] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J. M. Loingtier, and J. Irwin, "Aspect-oriented programming," in *Proceedings of the 11th European Conference on Object-Oriented Programming*, pp. 220–242, 1997.

[57] T. Elrad, M. Aksit, G. Kiczales, K. J. Lieberherr, and H. Ossher, "Discussing aspects of AOP," *Communications of the ACM*, vol. 44, no. 10, pp. 33–38, 2001.

[58] "Aspect-oriented programming with Spring." `http://docs.spring.io/spring/docs/2.0.x/reference/aop.html`. Accessed: 2013-11-04.

[59] "AspectJ." `http://eclipse.org/aspectj/`. Accessed: 2013-11-06.

[60] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. van Doorn, and P. Khosla, "Pioneer: Verifying code integrity and enforcing untampered code execution on legacy systems," *ACM SIGOPS Operating Systems Review*, vol. 39, no. 5, pp. 1–16, 2005.

[61] M. Bellare and C. Namprempre, "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm," *Journal of Cryptology*, vol. 21, no. 4, pp. 469–491, 2008.

[62] J. Katz and M. Yung, "Unforgeable encryption and chosen ciphertext secure modes of operation," in *Fast Software Encryption*, pp. 284–299, Springer, 2001.

[63] Y. Desmedt, "Man-in-the-middle attack," in *Encyclopedia of Cryptography and Security*, pp. 368–368, Springer, 2005.

[64] D. McGrew and J. Viega, "The Galois/Counter mode of operation (GCM)," *NIST Special Publication 800-38D*, 2004.

[65] Aspect Security, "Javasnoop." `https://www.aspectsecurity.com/research/appsec_tools/javasnoop/`. Accessed: 2013-11-05.

[66] H. J. La and S. D. Kim, "A conceptual framework for provisioning context-aware mobile cloud services," in *Proceedings of the 3rd IEEE International Conference on Cloud Computing (CLOUD'10)*, pp. 466–473, 2010.

[67] "Amazon EC2 instances." `http://aws.amazon.com/ec2/instance-types/#instance-details`. Accessed: 2013-11-04.

[68] I. Bate, G. Bernat, G. Murphy, and P. Puschner, "Low-level analysis of a portable Java byte code WCET analysis framework," in *Proceedings of the 7th IEEE International Conference on Real-Time Computing Systems and Applications*, pp. 39–46, 2000.

[69] J. Schiller and J. Peregoy, "Summary health statistics for US adults: National health interview survey," *Vital Health Statistics Series 10*, no. 256, 2011.

[70] US Department of Labor, Bureau of Labor Statistics, "Persons with a disability: Labor force characteristics - 2009." `http://www.bls.gov/news.release/archives/disabl_08252010.pdf`. Accessed: 2013-11-04.

[71] N. Giudice and G. Legge, "Blind navigation and the role of technology," in *The Engineering Handbook of Smart Technology for Aging, Disability and Independence*, pp. 479–500, John Wiley & Sons, 2008.

[72] R. W. Emerson and D. Sauerburger, "Detecting approaching vehicles at streets with no traffic control," *Journal of Visual Impairment & Blindness*, vol. 102, no. 12, pp. 747–760, 2008.

[73] J. Kwan and E. Tam, "Transportation services in Asia," in *The Engineering Handbook of Smart Technology for Aging, Disability and Independence*, pp. 546–566, John Wiley & Sons, 2008.

[74] "ADA accessibility guidelines for buildings and facilities, 2002." `http://www.adaag.com/ada-accessibility-guidelines/index.php`. Accessed: 2013-11-04.

[75] World Health Organization, "Fact sheet no 282: Visual impairment and blindness." `http://www.who.int/mediacentre/factsheets/fs282/en/`. Accessed: 2013-11-04.

[76] P. Gregor, A. F. Newell, and M. Zajicek, "Designing for dynamic diversity: Interfaces for older people," in *Proceedings of the 5th ACM International Conference on Assistive Technologies*, pp. 151–156, 2002.

[77] Google Inc., "Google glass." `http://www.google.com/glass/start/`. Accessed: 2013-11-07.

[78] R. de Charette and F. Nashashibi, "Real time visual traffic lights recognition based on spot light detection and adaptive traffic lights templates," in *Proceedings of the IEEE Intelligent Vehicles Symposium*, pp. 358–363, 2009.

[79] W. Crandall, J. Brabyn, B. L. Bentzen, and L. Myers, "Remote infrared signage evaluation for transit stations and intersections," *Journal of Rehabilitation Research and Development*, vol. 36, no. 4, pp. 341–355, 1999.

[80] V. Ivanchenko, J. Coughlan, and H. Shen, "Detecting and locating crosswalks using a camera phone," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW'08)*, pp. 1–8, 2008.

[81] Y. K. Kim, K. W. Kim, and X. Yang, "Real time traffic light recognition system for color vision deficiencies," in *Proceedings of the IEEE International Conference on Mechatronics and Automation (ICMA'07)*, pp. 76–81, 2007.

[82] T. Shioyama, H. Wu, N. Nakamura, and S. Kitawaki, "Measurement of the length of pedestrian crossings and detection of traffic lights from image data," *Measurement Science and Technology*, vol. 13, no. 9, pp. 1450–1457, 2002.

[83] S. Bohonos, A. Lee, A. Malik, C. Thai, and R. Manduchi, “Universal real-time navigational assistance (URNA): An urban bluetooth beacon for the blind,” in *Proceedings of the 1st ACM SIGMOBILE International Workshop on Systems and Networking Support for Healthcare and Assisted Living Environments*, pp. 83–88, 2007.

[84] “Speech-enabled eyes-free Android applications.” `http://eyes-free.googlecode.com`. Accessed: 2013-11-05.

[85] Y. Freund and R. Schapire, “A decision-theoretic generalization of on-line learning and an application to boosting,” *Journal of Computer and System Sciences*, vol. 55, pp. 119–139, 1997.

[86] R. Lienhart and J. Maydt, “An extended set of haar-like features for rapid object detection,” in *Proceedings of the IEEE International Conference on Image Processing*, vol. 1, pp. 900 – 903, 2002.

[87] “Google Maps.” `http://maps.google.com`. Accessed: 2013-11-05.

[88] J. Correa, E. Katz, P. Collins, and M. Griss. ”Room-level Wi-Fi location tracking,” Technical Report MRC-TR-2008-02, Carnegie Mellon University, November 2008.

[89] “Micello.” `http://www.micello.com/`. Accessed: 2013-11-07.

[90] T. McDaniel, S. Krishna, V. Balasubramanian, D. Colbry, and S. Panchanathan, “Using a haptic belt to convey non-verbal communication cues during social interactions to individuals who are blind,” in *Proceedings of the IEEE International Workshop on Haptic Audio Visual Environments and Games (HAVE’08)*, pp. 13–18, 2008.

[91] P. Viola and M. J. Jones, “Robust real-time face detection,” *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, 2004.

VITA

VITA

Pelin Angin received the Ph.D. degree in computer science from Purdue University under the supervision of Professor Bharat K. Bhargava. Her research interests lie mainly in the areas of mobile-cloud computing, information security and distributed systems.

She received the B.S. degree in computer engineering from Bilkent University, Ankara, Turkey in 2007. In the summer of 2010, she worked at the Lawrence Livermore National Laboratory as an intern. She received a Ross Fellowship in 2007, Purdue CETA Teaching Award and ACM Graduate Teaching Assistant Award in 2011, Raymond Boyce Graduate Teacher Award and Bilsland Dissertation Fellowship in 2012, and Phi Kappa Phi Love of Learning Award in 2013.