An Agent-based Optimization Framework for Mobile-Cloud Computing

Pelin Angin* and Bharat Bhargava
Purdue University
West Lafayette, Indiana, USA
{pangin, bb}@cs.purdue.edu

Abstract

The proliferation of cloud computing resources in the 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 between mobile and cloud resources, is expected to become increasingly popular in mobile application development. Dynamic partitioning of applications between mobile and cloud platforms based on resource availability is crucial in achieving the best performance for any computationally intensive mobile application. In this paper, we propose a dynamic performance optimization framework for mobile-cloud computing utilizing mobile agent-based application partitions. The proposed framework imposes minimal infrastructural requirements on the cloud servers, therefore exhibiting widespread applicability, as opposed to previous approaches with stricter requirements. Experiments with two real-world mobile applications serve as an initial feasibility study of the proposed framework and demonstrate the superiority of the proposed approach over monolithic execution of resource-intensive applications on mobile devices.

Keywords: Mobile-Cloud Computing, Optimization Framework, Agent

1 Introduction

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, computation-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. Cloud computing is already utilized by many mobile applications today. Current mobile applications mostly involve an inflexible split of computation between the mobile and cloud platforms, following the client-server paradigm with hard-coded 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

*Corresponding author: 305 N University St, West Lafayette, IN 47907 USA, Tel: +1-765-430-2140, Web: `http://web.ics.purdue.edu/~pangin/`

conditions. 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. In this paper, we propose a dynamic execution time optimization framework for mobile-cloud computing. 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 rest of this paper is organized as follows: Section 2 reviews related work in mobile-cloud computing; Section 3 provides an overview of mobile agents, which are the basic building blocks of the proposed framework; Section 4 describes the proposed computation framework in detail; Section 5 presents experiment results for the application of the proposed framework on two real-world mobile applications; and Section 6 concludes the paper with future work directions.

2 Related Work

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] designed 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] proposed 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] proposed 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. A recent proposal for securing cloud MANET applications [4] 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.

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 [5] and MAUI [6], 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. [7] propose a partitioning and execution framework specifically for mobile data stream applications. Huang et al. [8] 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. [9] propose an offloading framework limited to JavaScript based applications. The recently proposed ThinkAir [10] 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.

Li et al. [11] 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.

Zhang et al. [12] 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.

3 Mobile Agents Overview

3.1 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 [13]. 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 [13]:

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.

Figure 1 demonstrates the difference between the mobile agent paradigm and the client-server paradigm in terms of the communication involved upon the request of a client to a data server. As seen in the figure, while the client has to maintain a communication link with the service provider for the transfer of data in the client-server architecture, a mobile agent is shipped to the service provider and returns to the client only when it has collected all the requested data in the mobile agent paradigm.

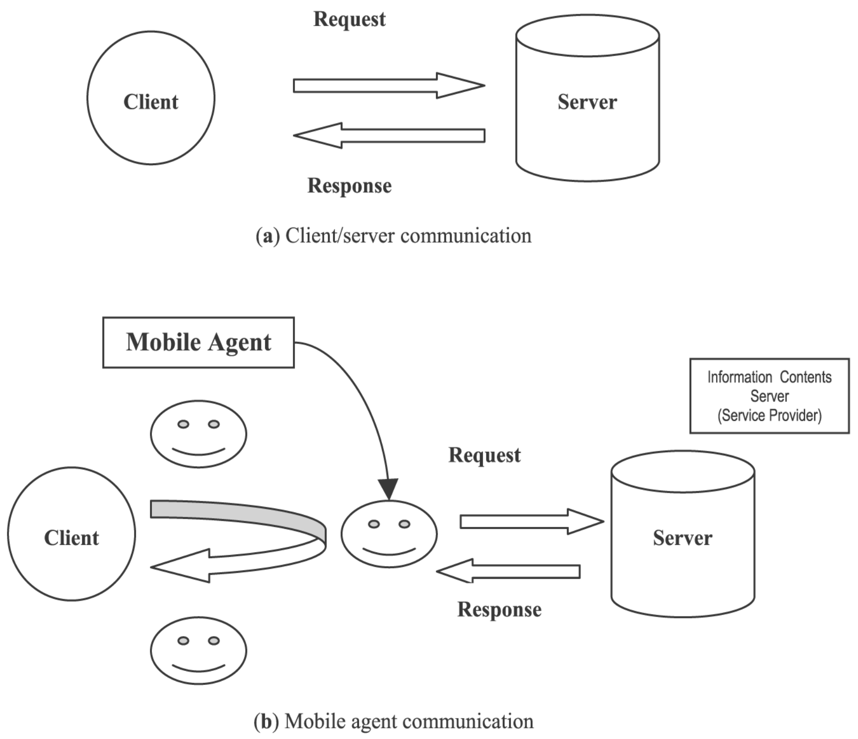

Request
Client
Server
Response
(a) Client/server communication
Mobile Agent
Information Contents Server (Service Provider)
Request
Client
Server
Response
(b) Mobile agent communication

Figure 1: Mobile agent vs. client-server communication [14].

3.2 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. The main advantages of mobile agent computing as listed by Chess et al. [13] making them effective tools for mobile-cloud computing are the following:

- *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 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.

3.3 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 [15]. 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 2 demonstrates the distributed JADE architecture.

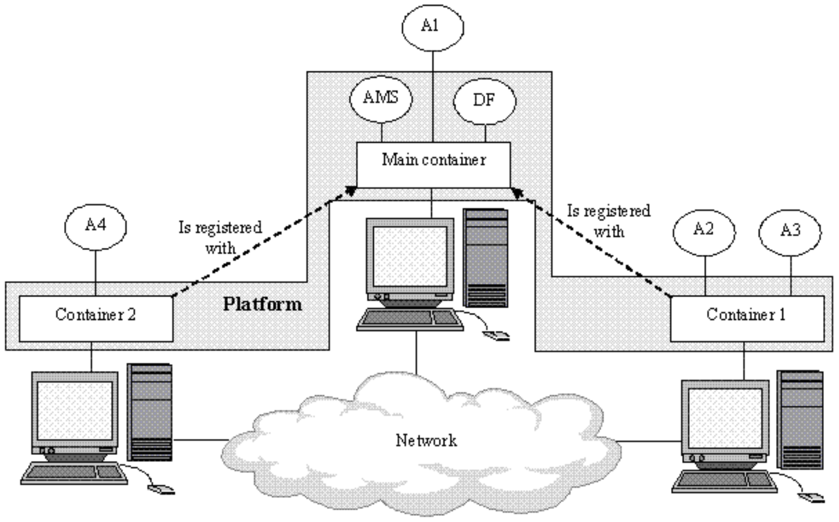


Figure 2: JADE distributed architecture [16].

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 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 [15].

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 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 [17].

4 Proposed Computation Framework

Figure 3 shows a high level view of the proposed computation framework. Each mobile application in the proposed framework consists of a set of agent-based application partitions that are offloadable to the cloud for execution (*P*2 and *P*3 in the figure), 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 (*P*1 in the figure). Partitioning of the application into these two types of components should be 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 [6].

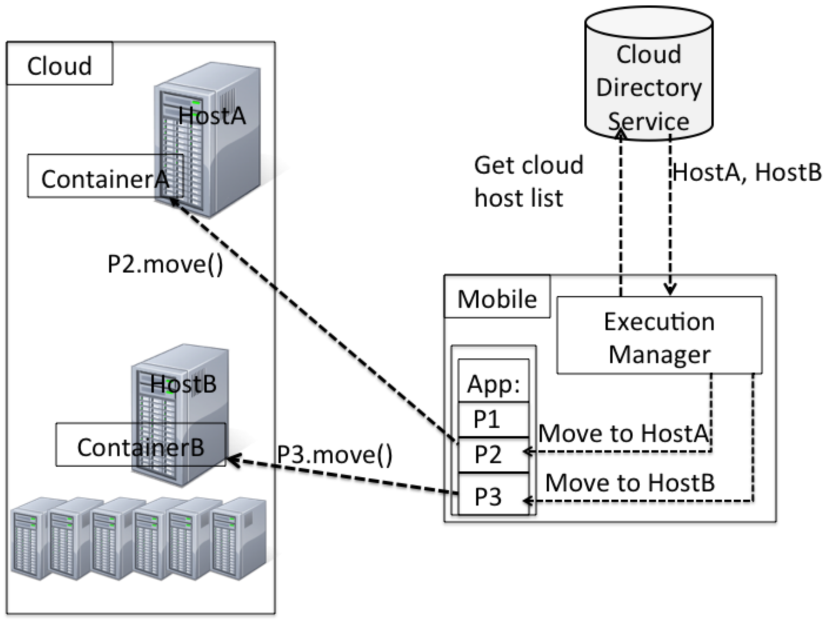


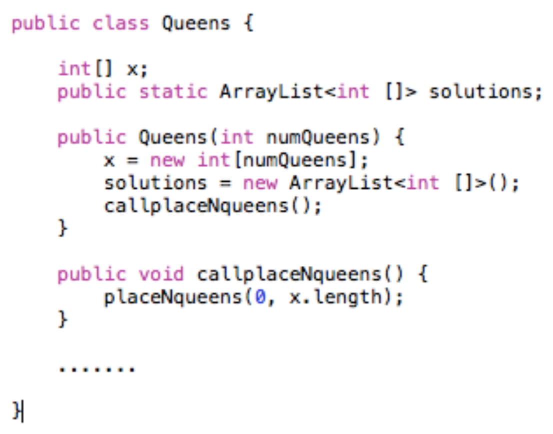
Figure 3: High level view of the proposed computation framework.

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 partitions is created by the execution manager. If the execution plan requires offloading a particular application partition, a bridge is formed between the caller of that partition and the *cloud host* selected by the execution manager, through which the offloaded partition migrates to the container in the host, carrying along its input parameters. Upon migration, the partition starts executing and communicates its output data to the caller through the same bridge. The four main components of the proposed framework are described below.

4.1 Agent-based Application Partitions

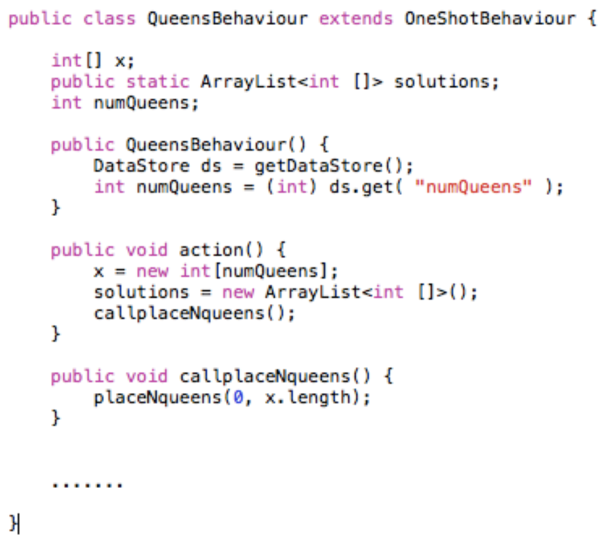
An agent-based application partition is a chunk of application code packed in a mobile agent, that is executable in a cloud host. Agent-based application partitions provide great advantages over existing mobile-cloud program partitioning techniques discussed in Section 2 due to their autonomous computing capabilities. Autonomy of these application partitions 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 partition to the corresponding agent-based partition is achieved through a *behaviour* as introduced in Section 3.3. In the current framework, application partitions 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). Figure 4a shows a sample application partition and Figure 4b shows the corresponding agent-based partition.



```
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);
    }

    .......

}
```

(a) Application partition for NQueens puzzle.



```
public class QueensBehaviour extends OneShotBehaviour {

    int[] x;
    public static ArrayList<int []> solutions;
    int numQueens;

    public QueensBehaviour() {
        DataStore ds = getDataStore();
        int numQueens = (int) ds.get( "numQueens" );
    }

    public void action() {
        x = new int[numQueens];
        solutions = new ArrayList<int []>();
        callplaceNqueens();
    }

    public void callplaceNqueens() {
        placeNqueens(0, x.length);
    }

    .......

}
```

(b) Corresponding agent-based partition.

Figure 4: Application partition transformation to an agent-based partition.

4.2 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. After choosing the most promising cloud hosts, the execution manager uses the following cost model to make offloading decisions for each offloadable application partition (Note that a sub-partition s of an application partition p is a partition invoked by p):

Let

td_p: time to execute application partition p on the device less the time to execute its sub-partitions

tc_p: time to execute application partition p wholly in the cloud with all of its sub-partitions

s_p: size of the data that is sent to partition p from its super-partition for a single invocation of p

n_p: the number of consecutive invocations of partition p by its super-partition

cd_p: the cost (in terms of execution time) of executing application partition p locally on the device
cc_p: the cost (in terms of execution time) of executing application partition p in the cloud
b: the network bandwidth available to the application
p_s: the set of sub-partitions of p.

Then cd_p and cc_p are calculated as follows:

$$cd_p = td_p + \sum_{i \in p_s} argmin(cd_i, cc_i) \tag{1}$$

$$cc_p = tc_p + s_p * n_p / b \tag{2}$$

In order to determine td_p and tc_p, 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.

When the execution manager is making decisions regarding the execution location of agent-based application partitions, it uses the following two heuristics:

- If a partitions is migrated, all of its sub-partitions (partitions called by this partition) will too: This is based on the assumption that the cloud hosts always have more computing power than the mobile device, i.e. the sub-partition execution time in the cloud will be shorter than on-device execution for every sub-partition. As the sub-partitions will involve data communication with the offloaded partition, keeping them together will prevent the penalty to be incurred by network latency during communication.

- Application partitions with frequent communication should be kept together if the time savings from offloading the sub-partition is below a certain threshold: Frequent communication between application partitions on different platforms could actually hurt performance under variable network conditions if the estimated time savings for executing the sub-partition in the cloud is low. Furthermore, there is a cost associated with processing (saving state etc.) due to offloading the sub-partition, which justifies the use of this heuristic.

4.3 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 latest interactions with instances can be provided by this service.

4.4 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 [18].

5 Experimental Evaluation

Experiments were performed with two real-world mobile applications to evaluate the performance of the proposed framework in terms of application execution time and energy consumption on the mobile device. A Motorola Atrix 4G device (with Dual-core 1GHz processor) [19] running the Android 2.3 [17] operating system was used to run the mobile applications and virtual machine instances in the east1-a region of the Amazon Elastic Compute Cloud (EC2) [18] were used as the cloud hosts. To measure the energy consumed by the applications due to CPU and Wi-Fi utilization, we used PowerTutor [20], a free energy measurement tool for Android devices. In all experiments, the relevant application was the only one running on the device in addition to the PowerTutor application. For the offloaded execution cases, a moderate speed Wi-Fi connection was used to send/receive data from the cloud servers and each offloaded partition was the only one executing on the chosen virtual machine instance (i.e. no multi-tenancy involved).

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

Figure 5 provides a comparison for device-only vs. offloaded execution time of the Sudoku solver component for different number 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.

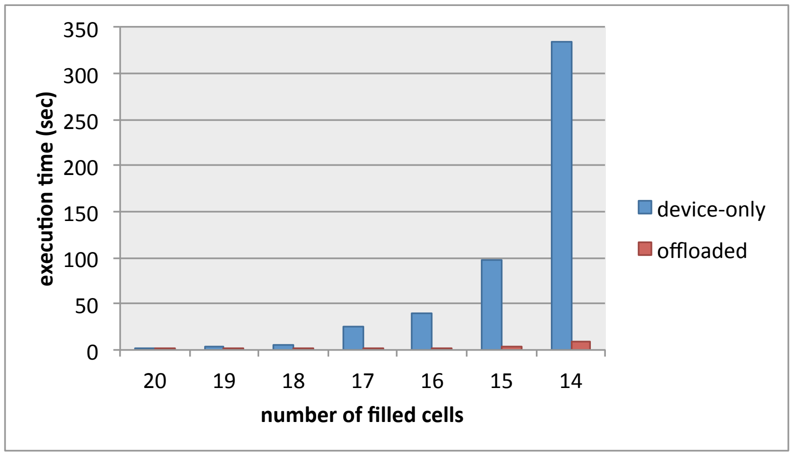


Figure 5: Sudoku solver execution time for device-only vs. offloaded execution.

Figure 6 shows the total (Wi-Fi + CPU) energy consumption on the mobile device for device-only vs. offloaded execution of the Sudoku solver. We see that offloaded execution achieves significantly higher

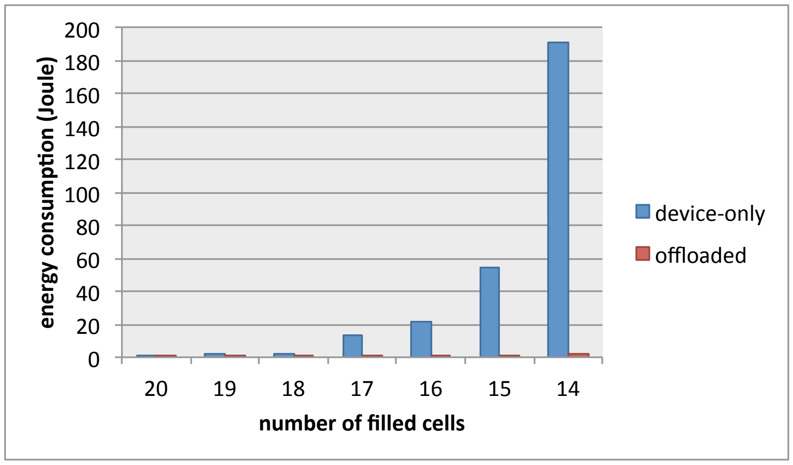


Figure 6: Sudoku solver energy consumption for device-only vs. offloaded execution.

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 7 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.

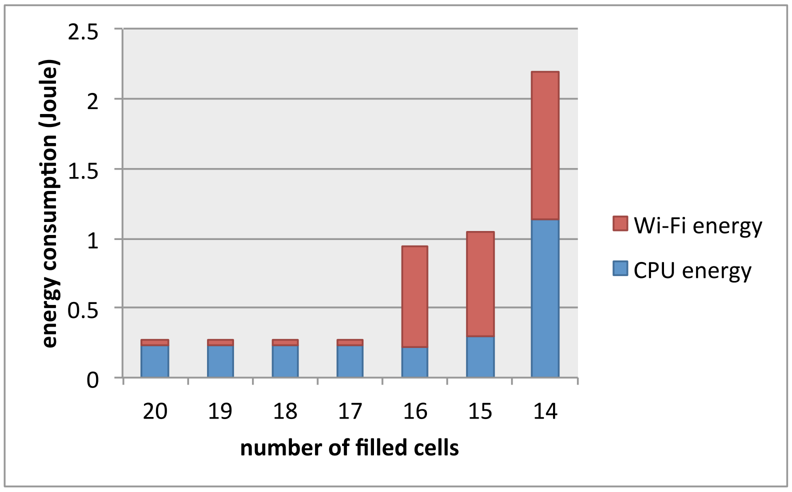


Figure 7: Sudoku solver energy components for offloaded execution.

5.2 Experiments with the 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).

5.2.1 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.

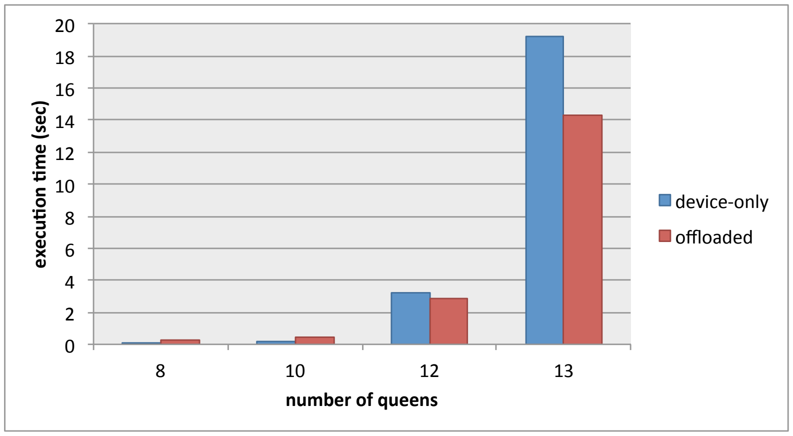


Figure 8: NQueens solver (all solutions) execution time for device-only vs. offloaded execution.

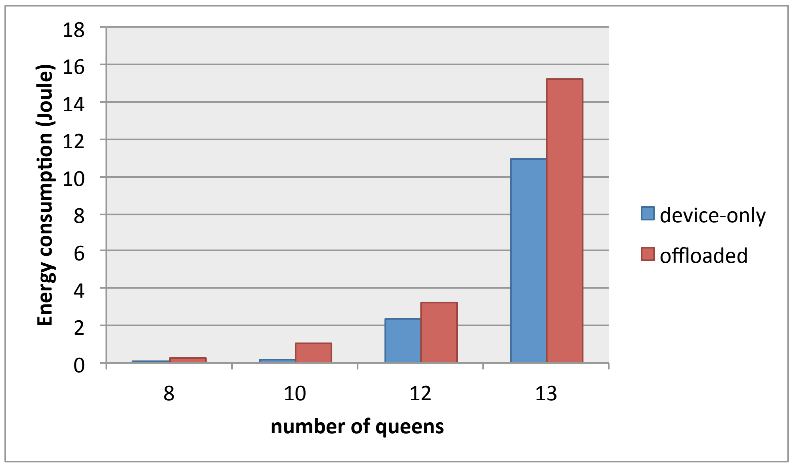


Figure 9: NQueens solver (all solutions) energy consumption for device-only vs. offloaded execution.

Figure 8 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 9 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.

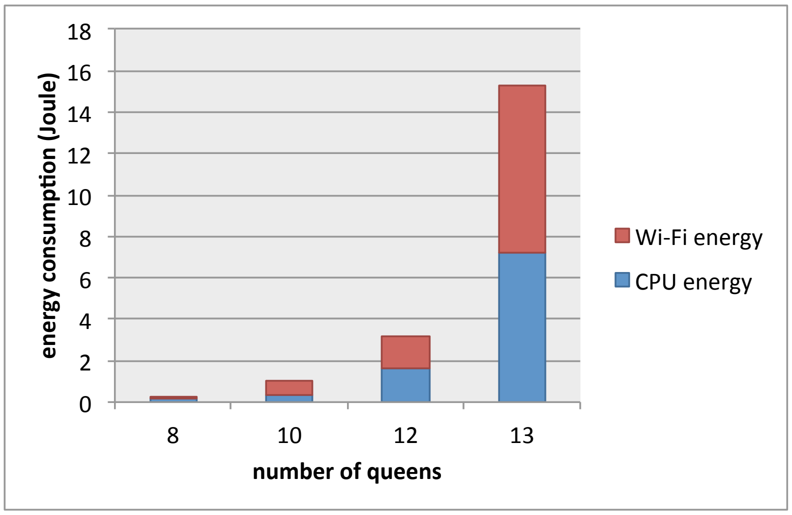


Figure 10: NQueens solver (all solutions) energy components for offloaded execution.

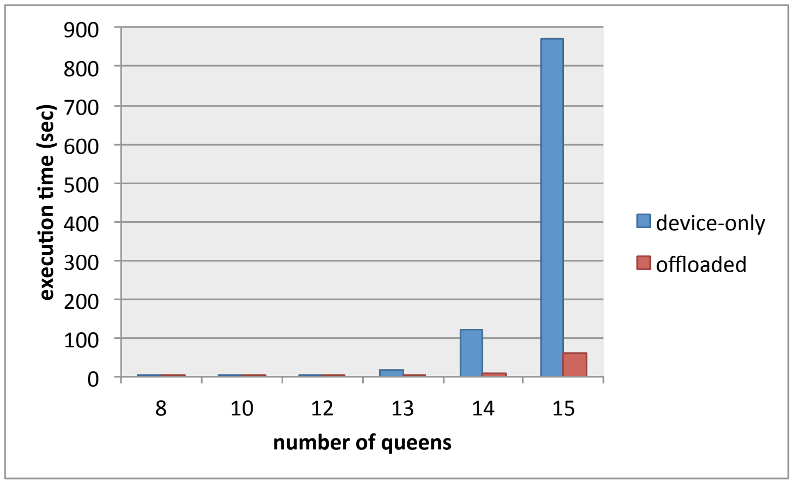


Figure 11: NQueens solver (number of solutions) execution time for device-only vs. offloaded execution.

Figure 10 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.

5.2.2 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 5.2.1, but only returns the number of solutions (i.e. puzzles are solved in place, not requiring memory allocation for each solution). Figure 11 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.

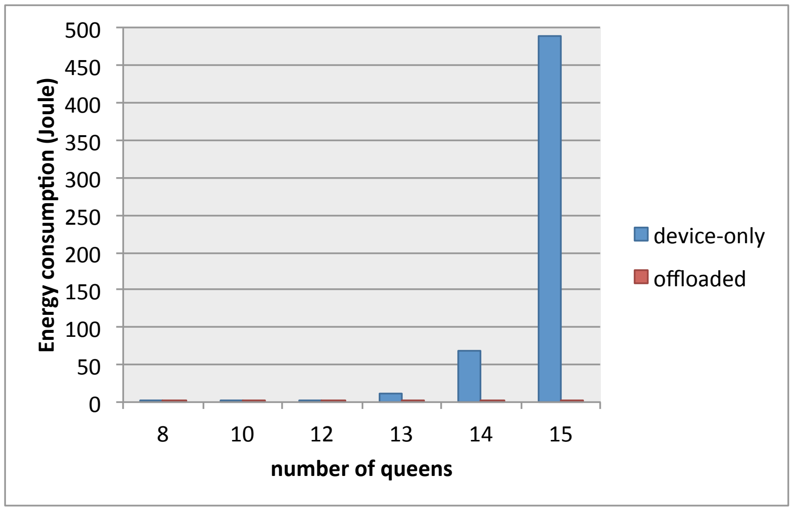


Figure 12: NQueens solver (number of solutions) energy consumption for device-only vs. offloaded execution.

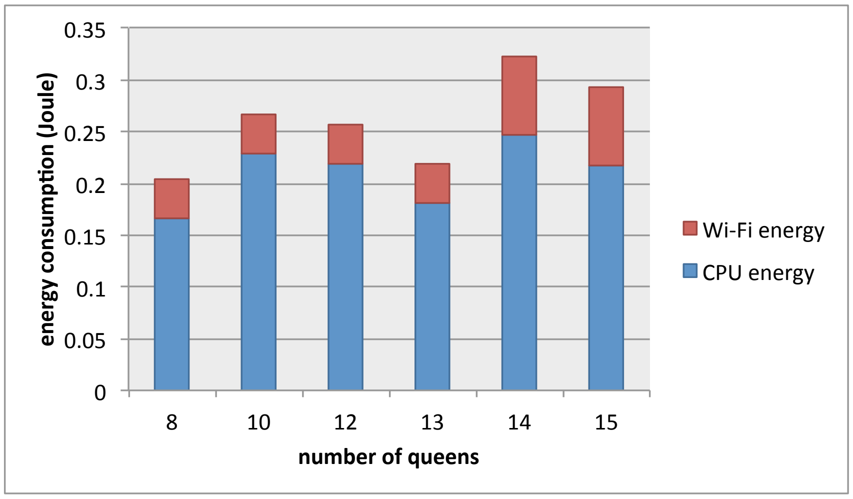


Figure 13: NQueens solver (number of solutions) energy components for offloaded execution.

Figure 12 shows a comparison of the total energy consumption on the mobile device for on-device vs. offloaded execution. The energy consumption for offloading is significantly lower than for on-device execution when the number of queens is more than 13. For 15 queens, offloading consumes about 1600 times less energy. The distribution of energy consumption over Wi-Fi and CPU utilization for offloaded execution is seen in Figure 13. As expected, Wi-Fi energy consumption is very low, as the response data contains only a single integer.

5.2.3 Effect of multi-threading

In this set of experiments, we evaluate the performance of a multi-threaded version of the NQueens puzzle from section 5.2.2, 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 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 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 2 shows that the total energy consumption on the mobile device is not affected by multi-threading.

Table 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

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 multi-threaded execution time. Specifications for the machine instance types we used in the experiments are shown in Table 3.

Table 3: Amazon EC2 instance type specifications [18]

Instance type	memory	number of cores	IO performance
Medium	3.75 GB	1	moderate
Large	7.5 GB	2	moderate
2x Large	30 GB	8	high

Figure 14 compares execution times for the offloaded 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.

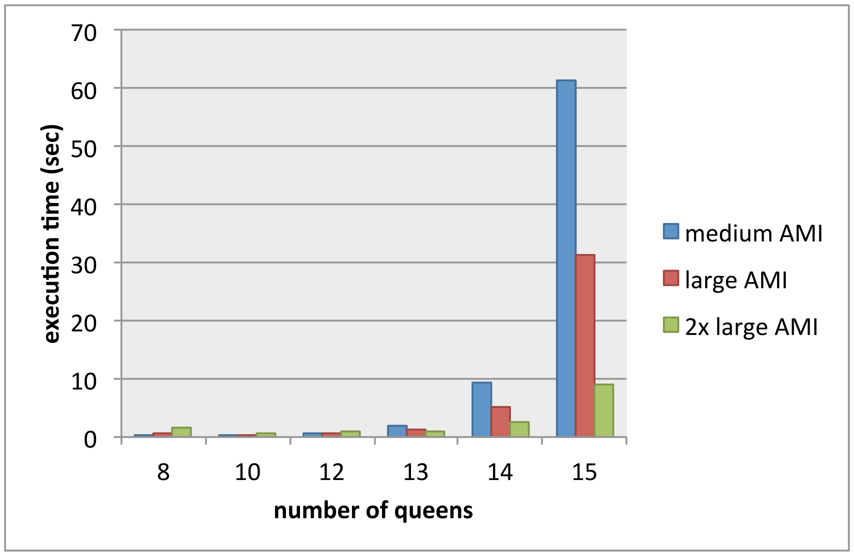


Figure 14: Multi-threaded NQueens solver offloaded execution time for different cloud host types.

6 Conclusion and Future Work

In this paper, we proposed a dynamic performance optimization framework for mobile-cloud computing using mobile agent based application partitions, imposing minimal structural requirements on the cloud. Experiments performed with two real-world applications demonstrate that the proposed framework is promising for improved performance and wide adoption in mobile-cloud computing. Future work will focus on the three main enhancements of the framework as described below:

- **Security**: One of the main problems that the wide adoption of cloud computing faces today is the security risks associated with using the cloud: 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. Our future work will involve extension of the current framework to ensure that the offloaded application partitions are tamper resistant, i.e. any tampering with the code or data of the agent-based partitions will be detected and upon tamper detection, the agent will be forced to move to a different platform.

- **Performance profiling**: A static performance profiler will fall short in capturing the performance for every possible problem size, as it could prove hard to enumerate all possible problem sizes for a specific program partition. Our future work will involve investigating and developing a performance profiler for the execution manager component of the framework, that is not strictly dependent on exact performance measurement values for various problem sizes.

- **Cloud resource provisioning**: The performance of an agent-based application partition on a public cloud host depends on the availability of resources on that host as well as the sharing of those resources with others. Even if a cloud host has very high computing power, simultaneous use of its resources by a large set of clients can significantly degrade the performance for each client to levels below acceptable for real-time tasks. Even though cloud resource monitors exist for some cloud platforms today, the information they provide may not be accurate for real-time tasks and such monitors may not always be accessible. In order to achieve accurate resource provisioning for cloud hosts, we plan to augment mobile agents with the capability to probe for specific resources on the host they migrate to. Once an agent is capable of resource provisioning, it can clone itself on multiple cloud hosts and a performance comparison for the different hosts can be performed to determine the most promising host for a specific application partition.

7 Acknowledgments

This research is supported by a generous grant from Amazon Web Services (http://aws.amazon.com).

References

[1] H. hua 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, no. 3, 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 *Proc. of the 1st IEEE International Conference on Pervasive Computing and Communications (PerCom'03), Fort Worth, Texas, USA*. IEEE, March 2003, pp. 107–114.

[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 *Proc. of the 9th IFIP/IEEE International Symposium on Integrated Network Management (IM'05), Nice, France*. IEEE, May 2005, pp. 3–16.

[4] D. Huang, X. Zhang, M. Kang, and J. Luo, "MobiCloud: Building secure cloud framework for mobile computing and communication," in *Proc. of the 5th IEEE International Symposium on Service Oriented System Engineering (SOSE'10), Nanjing, China*. IEEE, June 2010, pp. 27–34.

[5] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "CloneCloud: Elastic execution between mobile device and cloud," in *Proc. of the 6th European Conference on Computer Systems (EuroSys'11), Salzburg, Austria*. ACM, April 2011, pp. 301–314.

[6] E. Cuervo, A. Balasubramanian, D. ki Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: Making smartphones last longer with code offload," in *Proc. of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys'10), San Francisco, California, USA*. ACM, June 2010, pp. 49–62.

[7] 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 *Proc. of the 5th IEEE International Conference on Cloud Computing (CLOUD'12), Honolulu, Hawaii, USA*. IEEE, June 2012, pp. 794–802.

[8] D. Huang, P. Wang, and D. Niyato, "A dynamic offloading algorithm for mobile computing," *IEEE Transactions on Wireless Communications*, vol. 11, no. 6, pp. 1991–1995, June 2012.

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

[10] 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 *Proc. of the 31st IEEE International Conference on Computer Communications (INFOCOM'12), Orlando, Florida, USA*. IEEE, March 2012, pp. 945–953.

[11] X. Li, H. Zhang, and Y. Zhang, "Deploying mobile computation in cloud service," in *Proc. of the 1st International Conference on Cloud Computing (CloudCom'09), Beijing, China*. Springer-Verlag, December 2009, pp. 301–311.

[12] X. Zhang, J. Schiffman, S. Gibbs, A. Kunjithapa, and S. Jeong, "Securing elastic applications on mobile devices for cloud computing," in *Proc. of the ACM Workshop on Cloud Computing Security (CCSW'09), Chicago, Illinois, USA*. ACM, November 2009, pp. 127–134.

[13] D. Chess, C. Harrison, and A. Kershenbaum, "Mobile agents: Are they a good idea?" in *Mobile Object Systems Towards the Programmable Internet, LNCS*, J. Vitek and C. Tschudin, Eds. Springer-Verlag, 1997, vol. 1222.

[14] 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.

[15] T. I. Lab, "Java agent development framework," http://jade.tilab.com/, 2013.

[16] ——, "Wade," http://jade.tilab.com/wade/html/home-getting.htm, 2013.

[17] G. Inc., "Android," http://www.android.com, 2013.

[18] A. W. S. Inc., "Amazon elastic compute cloud," http://aws.amazon.com/ec2, 2013.

[19] M. M. LLC, "Motorola atrix 4g," http://www.motorola.com/us/consumers/Motorola-ATRIX-4G/72112,en_US,pd.html, 2013.

[20] 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/, 2013.

Pelin Angin is a Ph.D. Student at the Department of Computer Science at Purdue University. She received her BS degree in Computer Engineering at Bilkent University, Turkey in 2007. Her research interests lie in the fields of Mobile-Cloud Computing, Cloud Computing Security, Distributed Systems and Data Mining. She is currently working under the supervision of Professor Bharat Bhargava on dynamic computation partitioning between mobile and cloud platforms for performance optimization of real-time computationally intensive applications and the associated security issues.

Bharat Bhargava is a professor of the Department of Computer Science with a courtesy appointment in the School of Electrical and Computer Engineering at Purdue University. His research focuses on security and privacy issues in distributed systems. He is a Fellow of the Institute of Electrical and Electronics Engineers and of the Institute of Electronics and Telecommunication Engineers. In 1999, he received the IEEE Technical Achievement Award for his decade long contributions to foundations of adaptability in communication and distributed systems. He has been awarded the charter Gold Core Member distinction by the IEEE Computer Society for his distinguished service. He received Outstanding Instructor Awards from the Purdue chapter of the ACM in 1996 and 1998. In 2003, he was inducted in the Purdue's Book of Great Teachers. He serves on seven editorial boards of international journals. He also serves the IEEE Computer Society on Technical Achievement Award and Fellow committees. Professor Bhargava is the founder of the IEEE Symposium on Reliable Distributed Systems, IEEE Conference on Digital Libraries, and the ACM Conference on Information and Knowledge Management.