Contents lists available at ScienceDirect
Pervasive and Mobile Computing
journal homepage: www.elsevier.com/locate/pmc

Precise execution offloading for applications with dynamic behavior in mobile cloud computing

Yongin Kwon, Hayoon Yi, Donghyun Kwon, Seungjun Yang, Yeongpil Cho, Yunheung Paek*

Seoul National University, 1 Gwanak street, Seoul, Republic of Korea

A R T I C L E I N F O

Article history:
Received 29 September 2014
Received in revised form 4 September 2015
Accepted 1 October 2015
Available online xxxx

Keywords:
Mobile cloud computing
Execution offloading
Performance prediction

A B S T R A C T

In order to accommodate the high demand for performance in smartphones, mobile cloud computing techniques, which aim to enhance a smartphone's performance through utilizing powerful cloud servers, were suggested. Among such techniques, execution offloading, which migrates a thread between a mobile device and a server, is often employed. In such execution offloading techniques, it is typical to dynamically decide what code part is to be offloaded through decision making algorithms. In order to achieve optimal offloading performance, however, the gain and cost of offloading must be predicted accurately for such algorithms. Previous works did not try hard to do this because it is usually expensive to make an accurate prediction. Thus in this paper, we introduce novel techniques to automatically generate accurate and efficient method-wise performance predictors for mobile applications and empirically show they enhance the performance of offloading.

1. Introduction

Smartphones have become an essential part of a modern man's life, with around a billion devices activated worldwide for the Android platform alone. With its wide range of functions, such as GPS or cameras, and general purpose processors with gigabytes of storage, it has become natural to deploy more and more complex applications on smartphones. These applications, however, require a considerable amount of energy and computational power. As a result, users have to match the increasing computational complexity of applications with newer hardware. Yet they still suffer from limited battery lifetime all the same.

Mobile cloud computing, which utilizes cloud alongside mobile devices, is a promising approach to alleviate this problem. Within a mobile cloud computing framework, mobile devices do not need powerful hardware because most of the complicated computations are handled in the cloud. This approach extends battery lifetime, enables the use of the computation power of cloud systems, which typically exceeds even the newest mobile hardware, and lessens the need to upgrade user's devices. In recent years, techniques called *mobile execution offloading*, which is the act of transferring execution between smartphones and servers during run time, were proposed as a way of implementing mobile cloud computing. When an execution of a program thread on the smartphone gets to a certain point in its code, the thread is suspended and its current state for execution is packaged and shipped to a server. There, the thread is reconstructed from the shipped state and is resumed until it reaches the point to return, where it packages and transfers its state back to the smartphone. Finally, the original thread is updated by these states and is resumed.

* Corresponding author. Tel.: +82 28801748; fax: +82 28715974.
E-mail address: ypaek@sor.snu.ac.kr (Y. Paek).

http://dx.doi.org/10.1016/j.pmcj.2015.10.001
1574-1192/© 2015 Elsevier B.V. All rights reserved.

In ideal cases where the costs for state transfer and update can be neglected, any code region except for those using device resources like GPS or screens would benefit from remote execution. This is obvious because the server processor speed is much faster, and virtually no energy of the mobile device would be consumed while the thread runs on the server. In reality, however, the costs for state capturing and transferring may not be negligible and might even be a dominant factor that inhibits the regions from executing remotely. To mitigate the transfer cost, Yang et al. [1] dramatically reduced the size of transferred state by finding only the essential state needed to recreate a program on the server. Even with such efforts, however, the state transfer cost can still be high and inconstant in some cases, so offloading frameworks needed a way to selectively offload only when the code regions would benefit from the offloading.

It is for this reason that most mobile execution offloading frameworks implement a *dynamic code partitioning* module, which is also called the *solver*. The solver's key task is to determine which part of the program should be offloaded to the remote server for better performance by weighing the performance gains against the costs from the action of offloading at a certain point in the program. To accurately compare the gains with the costs, the solver should have an ability of predicting the program performance as precisely as possible before actual offloading is made. There have been several studies to build such solvers for their offloading frameworks. In most of the studies, they use the *history-based* prediction approach where they utilize the past profiled information as a basis for performance prediction of future runs [2–5]. For example in CloneCloud [2], they statically profile past information to make a set of decisions, called *scenarios*, which describe what code regions are to be offloaded at which runtime network condition (3G or WiFi). However, they have no regard for effects of inputs on program performance. In MAUI [3], they use the dynamically profiled information of a method as the predictor of future invocations. The history-based approach basically assumes that the program performance will be consistent regardless of the program input and environment. This assumption may hold for many applications, as empirically demonstrated by [3], in practice. However we have also found many other applications to which this does not apply because their performance is very *sensitive* to input values, that is, varying dynamically depending on the values.

To overcome the input-sensitivity problem of performance, in this work, we propose an alternative performance approach for execution offloading, which we call *feature-based* prediction. In this approach, we utilize as a basis for the prediction, the *features*, each of which characterizes a certain dynamic behavior of a program on a given set of input values. For example, the loop count can be a feature which represents how a loop behaves under the current input condition at runtime. To predict the program performance, we first go through the *off-line* phase where before actual program execution, we profile the execution behavior to establish a model which consists of a set of features that characterize a general behavior of the whole program execution. Then, in the *on-line* phase, we execute the program with real inputs, and whenever we need performance numbers of a specific region of the program for offloading, we extract the feature values for the current execution, which are in turn used to compute the output of the model, consequently representing the predicted performance of that part of the program.

We have implemented a feature-based prediction module for Android applications. To predict the program performance of input-sensitive applications, during the offline stage of the module, we execute an instrumented version of the applications on a set of training inputs which represent dynamic behaviors of them. The instrumented applications produce training outputs which include objective metrics of the program performance and the values of all possible features for each of the inputs. By using a machine learning technique [6] with the training output set, we select just a few among all possible features and build the model which is a function of (nonlinear combinations of) the features. Then, the prediction module provides our solver with a feature extractor and a model calculator to compute the feature values and the output of the model during the online stage. Finally, the solver makes an offloading decision with the output to improve the program performance. We show the impact of our work with three real applications, Chess Engine, Face Detection and Invaders. In our tests, we were able to reduce the execution time by up to 31.7% compared to previous methods. We also applied the prediction technique to an energy consumption problem. As a result, we could save the energy of smartphone by up to 57.2%. The rest of this paper is organized as follows: in Section 2, we first explain the basic concepts of execution offloading and then show how much impact prediction accuracy and global optimization have on offloading. Then in Section 3, we describe our techniques to precisely and efficiently predict various aspects of program performance for execution offloading. In Section 4, we describe our solver which utilizes our prediction techniques to offload our mobile code more precisely at runtime, attaining better performance of program execution. In Section 5, we experimentally demonstrate the effectiveness of our techniques which reduce significantly execution time or energy consumption. Finally, in Sections 6 and 7, we relate our work with others and conclude.

2. Background & motivation

In this section, we address how mobile execution offloading works and discuss how performance prediction accuracy and global optimization affect offloading precision.

2.1. Background

In order to evaluate the impact of prediction accuracy on the offloading performance, we present a simple offloading framework depicted in Fig. 1. The first step of the execution offloading is identifying which methods are remotely executable. There are two ways to identify the methods. The first is to use annotations within the source code to distinguish these