Journal of Network and Computer Applications 65 (2016) 12–35

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

An adaptive mobile cloud computing framework using a call graph based model

Mahir Kaya *, Altan Koçyiğit, P. Erhan Eren

Middle East Technical University, Informatics Institute, Middle East Technical University, Ankara, Turkey

ARTICLE INFO

Article history:
Received 1 July 2015
Received in revised form
22 December 2015
Accepted 19 February 2016
Available online 26 February 2016

Keywords:
Code offloading
Distribution transparency
Inversion of Control
Graph partitioning
Mobile cloud computing

ABSTRACT

The use of mobile applications and their functionality are increasing day by day but mobile devices are still inferior to ordinary computers in terms of memory and processor capacity. Furthermore, the rapid depletion of the mobile devices' energy is still a major problem. Performance and energy shortcomings of mobile devices can be improved by using surrogate or cloud computing technologies. In particular, computation and memory intensive real time applications would be efficiently run by utilizing the resources of a remote server. In this paper, a novel offloading framework based on the Inversion of Control mechanism is developed to overcome the shortcomings and limitations of the current offloading approaches published in the literature. The proposed offloading framework reduces the burden on programmers. It implements application partitioning and code offloading via remote proxy classes and seamlessly provides callback functionality. In an application, it is possible to migrate different combinations of application components to remote servers. Some of these combinations can be productive and others can be counterproductive for offloading. In order to decide on components to be offloaded, a call graph model based on the collaboration of application's classes is developed. An offloading decision algorithm is presented to determine the classes to be offloaded by finding the best application partitioning in the graph. The framework and the graph model are evaluated by several experiments. Experimental results show that the proposed graph model fits well to the application partitioning problem. It has also been shown that offloading the optimal combination of components to remote servers can considerably reduce the execution time and energy consumption of mobile devices.

1. Introduction

Recent developments in communication and mobile computing technologies increase the demand for the transparent migration of computation intensive applications to resourceful servers. Since mobile devices (smartphones, tablets) still cannot compete against their desktop equivalents for resource intensive applications such as image and video processing, object recognition and augmented reality applications, their capabilities can be augmented by migrating CPU or memory intensive parts of the application to remote servers. In order to overcome the resource limitations of mobile devices, personal computers residing in the LAN to which the mobile device is currently attached or shared resources of the nearby cloudlet or remote cloud can be utilized for offloading. Offloading is an invaluable contribution since it is utilized in pervasive computing environments in order to augment the capability of resource-constrained mobile devices by migrating the components of applications such as classes, objects, services or methods to servers that are nearby machines (called surrogates) or the virtual machines of the cloud (Ou et al., 2007).

Mobile devices generally use two methods to benefit from a nearby powerful computer or cloud computing infrastructure. In the first method, a virtual machine (VM) is entirely moved to the remote server, re-started, made ready and returned the result after making the calculations. In this method, not only the network cost is too expensive, but also problems occur during instant calculations when smartphone sensors need to be used on demand. The second method is the application partitioning mechanism. This method can be grouped under three sub-categories; the proxy-based methods via the Remote Method Invocation (RMI) (Oracle 2010a), preparing computation intensive parts as a service using the Interface Definition Language (IDL) (Android 2008), and the OSGi service-based method (OSGi 2010). Partitioning an application and sending the offloadable components to remote servers incurs less overhead, but requires various degrees of program restructuring; and in both cases, problems can arise when an application runs processes that are dependent on the resources of a smartphone.

* Corresponding author at: Informatics Institute, Middle East Technical University, Ankara, Turkey.
E-mail addresses: kmahir@metu.edu.tr, mahirkaya@gmail.com (M. Kaya), kocyigit@metu.edu.tr (A. Koçyiğit), ereren@metu.edu.tr (P.E. Eren).

http://dx.doi.org/10.1016/j.jnca.2016.02.013
1084-8045/

Recent studies based on services (application components) that require the use of IDL and an OSGI middleware implement the computation intensive parts as services and migrate these services to a remote server. However, these services should be independent of resources of the smartphone such as sensors and cameras, which decrease the distribution transparency of the frameworks. Another problem is that even if services run locally, the application communicates with these services using Inter-Process Communication (IPC) via the network stack, which is time-consuming and thus limits the goal of the computation offloading in terms of increasing the performance and reducing the energy consumption. Marshalling (serialization) arguments of the services also creates argument inconsistencies if the remote server changes the fields of the method argument object. In addition, Microsoft's DCOM (Chung et al., 1998; Plášil and Stal 1998) and OMG's CORBA (Chung et al., 1998; Plášil and Stal 1998) are well-known architectures for distributed applications. However, they also suffer from argument inconsistency. It is assumed that arguments are passed by value or they are immutable.

Although significant research has been conducted on the mobile cloud computing systems (Ou et al., 2007; Cuervo et al., 2010; Kristensen and Bouvin 2010; Chun et al., 2011; Chen et al., 2012a; Kemp et al., 2012; Verbelen et al., 2012; Flores et al., 2015) there are still several challenges to be addressed as stated above concerning the design and implementation of a widely adopted framework and the selection of components to offload in current smartphone applications. The limited bandwidth in wireless networks as well as high and changing network latencies in a WAN environment also need to be considered (Satyanarayanan et al., 2009). Therefore, an adaptive, seamless offloading strategy should be implemented without resulting in any extra overhead for the smartphone applications.

The offloading technique presented in this study is based on the Inversion of Control (IoC) (Fowler 2004; Kaya et al., 2014). This technique seamlessly synchronizes resource access on both the smartphone and the surrogate/cloud side of an application and eliminates the limitations of the existing offloading approaches (Kumar et al., 2013; Shiraz et al., 2015). First, we address the software modules that are not suitable for offloading; such as software modules providing the Graphical User Interface (GUI), sensors and network components. Other components are candidates to be offloaded to resourceful servers but we also consider the components that may require access to smartphone resources; such as local database and sensors to provide a transparent callback functionality. The selection of the components to be offloaded depends on certain factors; such as the amount of the computation and data required for the method call, and the bandwidth of the wireless access network.

In this paper, in addition to the offloading technique, a framework is presented to simplify developing elastic applications with offloadable components and making dynamic offloading decisions for optimal application partitioning. To this end, we propose a novel graph model to collect the profiling information and then to identify the classes of the application to be offloaded and to be executed on the remote server. Constructing the call graph, the offloading decision problem is converted to the graph partitioning (min-cut) problem (Karypis and Kumar 1998). Finding an optimal solution for the graph partition is shown to be NP-Hard (Kernighan and Lin 1970; Karypis and Kumar 1998), therefore in this study, we implement a well-known graph partitioning heuristic, Fiduccia and Mattheyses (FM) heuristic (Fiduccia and Mattheyses 1982; Hendrickson and Leland 1995), to determine the minimum edge-cut that is the best offloading decision. We offload different combinations of application classes to identify the cases where offloading can be counterproductive. This is important in terms of finding an optimal solution for offloading to decrease the network cost involved.

The key contributions of this paper are; (1) providing distribution transparency for mobile cloud computing and (2) creating a graph model based on the method calls and using this model to decide on the best partition containing classes to be offloaded. Distribution transparency is achieved by creating proxies for offloadable classes both on the smartphone and on the server side using the offloading factory to achieve remote object access. The object ids are exchanged instead of the serialized objects across the network hence the objects in different devices collaborate with each other seamlessly. Offloading can be counterproductive if one of the offloaded classes of the application is heavily dependent on smartphone resources, which increases the network cost. Our graph model handles this issue by providing profitable application partitions for offloading.

To evaluate the offloading framework, we used the Android Operating System (OS) on the smartphone and J2SE on the server side, and assessed the performance of offloading on real time applications. An Object Recognition (OR) (Kemp et al., 2009) application was used since it shows the effectiveness of the proposed framework. Using the OR application, the offloading technique and decision model were implemented. The graph model allowed us to determine the best offloading decision and the results were validated conducting several experiments on the real time application. The second application is an image filter application that applies different image filtering algorithms.

This paper is organized as follows: Section 2 presents an overview of the related work on offloading methods and offloading decisions. The offloading approaches are summarized in Section 3. Section 4 describes the proposed graph model and offloading decision algorithm. Section 5 presents the offloading framework architecture and method calls with dynamic proxies and provides a pseudo-code of the offloading technique. Section 6 shows how the proposed offloading framework is implemented, and Section 7 presents the performance evaluation of the framework. Finally, Section 8 concludes the present work discussing the limitations of our study and possible future work.

2. Related work

In this section, we present an overview of the research in the literature regarding application partitioning and computation offloading. For application partitioning, graph based approaches can be used to partition an application graph into a number of disjoint subsets that contain the list of classes to be offloaded to resourceful servers. The cumulative weight of edges whose incident vertices are located in different virtual machines is called the edge-cut of the partition. The main goal of application partitioning is to minimize the edge-cut that determines the network cost. Although determining the optimal partitioning to distribute the components of applications is an NP-Hard problem, there are various classical heuristics offering solutions (Kernighan and Lin 1970; Fiduccia and Mattheyses 1982; Hendrickson and Leland 1995; Karypis and Kumar 1998); however, these heuristics need to be adapted to mobile environment in terms of costs and balance constraints. In addition, Ou et al. (2007) proposed a different multi-level graph partitioning heuristic for mobile applications coarsening the graph based on the heavy edge-light vertex algorithm. The coarsening phase continues until the subsets/partitions that are suitable for component distribution are achieved. The algorithm randomly chooses vertices and merges them with their neighbors that have low vertex cost and high edge weight. The edge weight of this heuristic is the frequency of the method call among different classes. The vertex weight represents memory and CPU processing cost. The runtime complexity of this heuristic is $O(|V|^3)$ ($|V|$ is the number of vertices) and the computation process for finding suitable partitions to offload is also expensive. Abebe and Ryan (2012) implemented the same heuristic but maintained the distributed partitions on the cloud side to decrease the memory-related costs.