Journal of Network and Computer Applications 40 (2014) 105–115

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

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

Design and evaluation of mobile offloading system for web-centric devices ☆

Sehoon Park [a], Qichen Chen [b], Hyuck Han [a,*], Heon Y. Yeom [b]

[a] Samsung Electronics, Gyeonggi-do, South Korea
[b] School of Computer Science and Engineering, Seoul National University, Seoul, South Korea

ARTICLE INFO

Article history:
Received 7 January 2013
Received in revised form
7 June 2013
Accepted 25 August 2013
Available online 13 September 2013

Keywords:
Offloading
Mobile device
JavaScript
Power saving
Platform-independent

ABSTRACT

Increasingly, smartphones are becoming one of the most popular mobile devices in the personal computing environment. As the need for a variety of mobile applications increases, the target mobile platform is a primary concern for mobile application developers. To reduce design complexity for different platforms and enhance the compatibility of applications on various mobile OSes, a JavaScript-based web environment became a main target framework for smartphone applications. Two particular characteristics of a smartphone are restricted power supply and low-end hardware resources, compared to high-end servers. Computing-intensive and rich graphics-based applications in a smartphone may fully utilize the CPU and consume a large amount of the battery power accordingly.

In this paper, we propose a platform-independent mobile offloading system, which is a delegated system for a web centric devices environment. Our offloading architecture uses a built-in proxy system that splits the original JavaScript-based application codes into the following two parts: a lightweight code for the mobile client and a computationally heavy code that runs on the server system. We adopt one of the web applications utilizing a combinatory search for our case study. Our evaluation shows that our mobile offloading system reduces the response time of the application running in the web browser and enables a high workload application to run on relatively low-end mobile devices. In addition, this method reduces power consumption of the device. Therefore, this web-based offloading architecture creates a new mobile computing environment and supports various OS platforms of mobile clients.

1. Introduction

The emergence of mobile devices has changed many aspects of the current personal computing environment. Various mobile platforms are rapidly introduced to meet our computing needs. The increasing speed of mobile networks, including W-CDMA, LTE, and IEEE 802.11n, accelerates the performance of mobile devices, and it also minimizes the overhead of the client–server model in the network environment. Mobile applications are becoming more elaborate in order to support complex applications, such as games that incorporate augmented reality and high-level mathematical computations. However, along with the complicated and rich graphic-intensive characteristic, these applications are still restricted to the hardware resources in the mobile computing environment.

As a solution to this issue, several research projects (Chun et al., 2011; Cuervo et al., 2010) propose offloading systems. However, their offloading functions rely on operating systems of mobile devices, and this may lead to non-trivial development overhead in terms of changes or upgrades of operating or runtime systems (detailed in Section 2). Moreover, there are many operating systems, such as Android (Google), iOS (Apple), RIM (BlackBerry), Symbian (Nokia), and Bada (Samsung), and this diversity causes major concern to mobile web application developers. Thus, mobile service providers or mobile manufacturers need platform-independent offloading systems. To enhance the compatibility of the mobile platform, many applications are integrated into a web browser, which is a common environment among the different mobile OS platforms. As a next-generation web standard, JavaScript has been represented as a universal platform language for mobile applications that run on web browsers. The emergence of HTML5 also derives from the need for a common platform for the mobile device, which mainly uses JavaScript-based platforms. We propose a platform-independent JavaScript offloading system to minimize the limitation of hardware resources, and to maximize the compatibility of various mobile OS platforms.

Heavy computational models (Hill et al., 2010; Lee et al., 2009) or gaming applications based on JavaScript require high CPU utilization, which shortens the battery lifetime. These applications usually require high-level hardware specifications and increase

☆A preliminary version of this paper was presented as a short paper at IEEE CCNC 2013.

* Corresponding author. Tel.: +82 31 325 3863.
E-mail addresses: tsh.park@samsung.com (S. Park), charliecqc@dcslab.snu.ac.kr (Q. Chen), hhyuck@dcslab.snu.ac.kr, hyuck.han@samsung.com (H. Han), yeom@snu.ac.kr (H.Y. Yeom).

1084-8045/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.jnca.2013.08.006

the cost of the devices, which is challenging for low-end devices. Low-cost smartphones are becoming popular in emerging markets, especially in developing countries. These devices reduce the cost and extend the battery lifetime. However, the main problem with them is that their capacity is limited to utilizing complicated web resources, such as a rich UI or highly computational JavaScript applications. To address the issue of the limited resources of low-end devices, we propose a new architecture for application offloading, in which a mobile client out-sources JavaScript functions to an offloading server. With our offloading technique, the low-end devices are able to support complicated UI and high computational web resources.

The basic idea behind our offloading system is to exploit the separation of heavy computation functions specified by the programmer's annotation. The computational parts are moved to the server as callee functions, while the client receives the lightweight caller functions. Our offloading methodology runs on the application (browser) and (source) code level; thus it is not limited to the platforms or runtime of the mobile OS. This offloading system reduces the level of CPU utilization and energy consumption of mobile devices, since the main computational works are offloaded to the server side. This client–server model may require a data traffic over the network as exchanging parameters between callee and caller functions, however the total amount of network traffic received by the client is not as big as the non-offloading method. This is because that the offloading functions will be excluded from the original source codes, and also the caller function will share only a small amount of parameters with a callee in our offloading model. The client device sends a few parameters for the computational function to the offloading server in our system, and receives a result only. Thus, the overhead is relatively small as compared to the other system which usually needs whole parts of functions or clones of the original source codes. Again, our offloading can easily detect the offloading functions by the annotation from a programmer, and it leads the fast separation of a caller and callee. Therefore, our annotation-based offloading system increases the performance of mobile clients. Also our platform-independent offloading engine based on the application level enlarges the compatibility of the various mobile platforms.

The key contribution of our paper is a new offloading methodology that increases the performance of mobile devices by reducing CPU utilization and saving energy. To evaluate our methodology, we first implement our offloading system, and we perform experiments by using a widely known JavaScript application from the Internet.

The rest of the paper is organized as follows: Section 2 reviews the background and related work on the offloading systems for mobile environments. Section 3 describes the design and implementation of our mobile offloading system. Section 4 presents a case study of our offloading system based on a casual game application in JavaScript. Section 5 presents an evaluation of the performance using a casual game in our mobile architecture, and Section 6 concludes our work.

2. Background and related work

There are several major distinguished research projects on the issues of computational power and the cost of CPU utilization in mobile computing environments. One is the CloneCloud (Chun et al., 2011) project, which uses nearby computers or data centers to speed up smartphone applications and reduces CPU utilization on mobile devices by cloning the entire image of a mobile device via cloud computing. This method allows mobile devices to be more power-efficient and reliable, but it requires pre-processing on the client side, which can increase the cost of mobile devices.

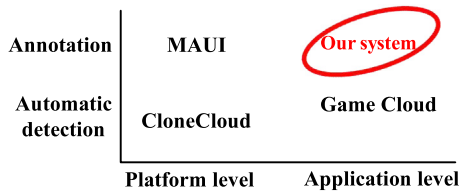

Fig. 1. Our system approach.

It can also lead to excessive network traffic from the cloud network to the device. Issues pertaining to security cannot be neglected either, as this architecture clones the complete image of a mobile system, include the user's private files, in third-party cloud storage over the network.

Another project is MAUI (Cuervo et al., 2010), which provides a fine-grained code offload based on the annotations by a programmer to maximize energy savings. This code determines at the runtime level which methods should be remotely executed as driven by an optimization engine under current connectivity constraints of the mobile device. However, experimental results have shown that this approach works only on Microsoft Windows platforms. Moreover, the level of the code offloading decision is based on the . NET framework. Therefore, this method is non-scalable and is not commonly adoptable on various mobile computing OS platforms. However, our approach is reliable, concise, and platform-independent, as it operates on JavaScript platform, which runs at the browser level.

Recently, LG U-plus, one of Korea's largest telecom companies, has launched a cloud platform (LG U+, 2012) for games on which users can access games that can be played on smartphones and personal computers on the cloud. The platform will store games provided by developers and publishers for customers who do not need to download them, but simply log in and play via its streaming services. This approach still requires a massive amount of network storage, and it is somewhat limited in its ability to be widely adapted on all mobile applications compared to our system. Figure 1 summarizes the various offloading architectures and emphasizes our offloading approach as very effective in terms of platform compatibility and design simplicity.

In automatic detection schemes for offloading systems such as Diaconescu et al. (2005), Xian et al. (2007), Chun et al. (2011), it is very important to design partitioning and offloading schemes for applications since offloading incurs network overhead from transmission delay and cost. Static partitioning such as Diaconescu et al. (2005) is not suitable if there are frequent bandwidth fluctuations in the mobile communication. Dynamic partitioning such as Xian et al. (2007), Chun et al. (2011) results in high overhead due to continuously determining partition. In Niu et al. (2013), authors proposed the Weighted Object Relational Graphs (WORG) to improve static partitioning and avoid high overhead of dynamic partitioning. In this study we focus on platform-independent offloading systems rather than partitioning strategies. Our system employs the application-level partition that is based on programmers' annotations to build the platform-independent offloading system. Therefore, if mobile web standards deals with QoS information such as bandwidth, it is possible to build platform-independent offloading systems with bandwidth-adaptive partitioning to maximize benefits of offloading. We remain this as future work.

In Chen et al. (2004), the authors suggest several guidelines to determine whether a function or component is considered for remote execution. The followings are key rules suggested: (i) A component that accesses local I/O devices including storage devices, a camera, and a GPS device should not be offloaded.