Offloading Guidelines for Augmented Reality Applications on Wearable Devices

Bowen Shi, Ji Yang[*], Zhanpeng Huang, Pan Hui
Department of Computer Science and Engineering,
The Hong Kong University of Science and Technology, Hong Kong
{bshiab, jyangaa}@connect.ust.hk, soaroc@ust.hk, panhui@cse.ust.hk

ABSTRACT

As Augmented Reality (AR) gets popular on wearable devices such as Google Glass, various AR applications have been developed by leveraging synergetic benefits beyond the single technologies. However, the poor computational capability and limited power capacity of current wearable devices degrade runtime performance and sustainability. Computational offloading strategy has been proposed to outsource computation to remote cloud for improving performance. Nevertheless, comparing with mobile devices, the wearable devices have their specific limitations, which induce additional problems and require new thoughts of computational offloading. In this paper, we propose several guidelines of computational offloading for AR applications on wearable devices based on our practical experiences of designing and developing AR applications on Google Glass. The guidelines have been adopted and proved by our application prototypes.

Categories and Subject Descriptors

H.5.1 [**Multimedia Information Systems**]: Augmented and virtual realities

General Terms

Design

Keywords

Offloading; Augmented reality; Wearable devices

1. INTRODUCTION

Nowadays, mobile devices, especially wearable devices such as Google Glass, have rapidly gained their popularity. In consequence, mobile computing and wearable computing have been attracting more attentions accompanied with various applications such as AR applications. Nevertheless, AR applications involve complex algorithms which require fast real-time processing and computation capability with power efficiency. Code offloading is one solution which utilizes nearby mobile devices and remote cloud for outstanding computations. A large amount of works [1, 2, 5, 7] have focused on designing offloading systems for mobile devices. However, wearable devices have their specific limitations. The computational capability, memory storage, and power capacity are more limited compared with mobile devices as illustrated in Table 1.

Furthermore, wearable devices do not have keyboard or mouse inputs, so they usually take advantage of audio input and object or gesture recognition instead. These techniques require additional computations and lead more power consumption compared with traditional mobile devices. Even with the rapid development of hardware technology, hardware equipped on wearable devices is still too limited to meet the demand of compute-intensive applications.

Traditional offloading mainly outsources computation to remote cloud. It improves applications performance on mo-

Category	Product	CPU	Memory	Battery
Wearable glass	Google Glass	OMAP 4430 SoC, dual-core @ 1GHz	1GB	570mAh
	VUZIX M100	OMAP4460 @ 1.2GHz	1G	600mAh
	Epson Moverio BT-200	1.2GHz dual-core TI OMAP 4480 CPU	1G	2720mAh[#]
	Meta Pro	i5 CPU	4G	1200mAh
	Icis Laforge	1GHz	500M	500mAh
Mobile device	Motorola s6	Quad-core 2.65GHz Krait	3G	3220mAh
	Samsung Galaxy S5	Quad-core 2.5GHz Krait 400 2G	2G	2800mAh
	Google Nexus 6	Quad-core 2.7GHz Snapdragon 3G	3G	3220mAh

[#] Wired to a standalone control box

Table 1: Hardware comparison of mobile devices and wearable AR glasses in current market

[*]Contact author

MM'15, October 26–30, 2015, Brisbane, Australia.
© 2015 ACM. ISBN 978-1-4503-3459-4/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2733373.2806402.

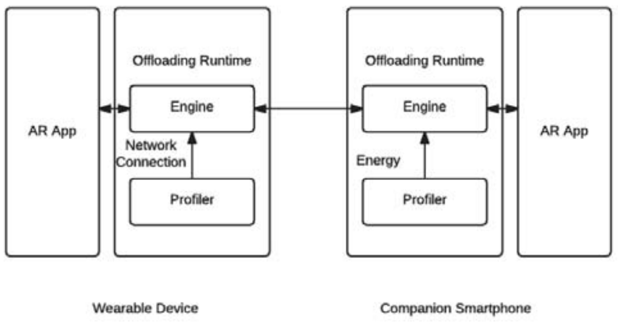

Figure 1: A simplified model of the adapted offloading system for AR applications on wearable devices

bile devices. However, the performance degenerates due to higher delay and possible failure of network connection. Since wearable devices usually work with companion mobile devices, it is possible to offload the computational tasks from wearable devices to the companion mobile devices.

In this paper, we propose several guidelines of computational offloading of AR application on wearable devices. These guidelines are designed based on our experience of designing and developing three AR applications on Google Glass. In the next section we will give detailed expatiation on these guidelines, followed by evaluation on the AR applications we have developed.

2. DESIGN GUIDELINES

In this section we present several guidelines for designing an offloading system specialized for AR applications on wearable devices. One of the most practical applications of offloading system is the Gabriel [6], a system for wearable cognitive assistance devices. In Gabriel, the so-called cloudlet is employed as an architecture element which is a 3-tier hierarchy: device-cloudlet-cloud. A cloudlet aims to bring the cloud closer. It can be a laptop or a notebook which the users usually carry with themselves. The applications on target device can be offloaded to either cloud or cloudlet. In cases when cloud is unavailable, the cloudlet will serve as the small data center to accelerate computing for wearable devices. Other work like Honeybee [4] and Misco [3] aims to design programming frameworks for smartphone device-to-device (D2D) offloading.

At present, the smartphone has become a powerful device, which is reliable enough to work as a data processing hub for offloaded tasks. In addition, users tend to carry smartphone meanwhile they take wearable device, so they can work as a pair. As the wearable device and smartphone are usually close to each other, there will be little network connection problem. In our system, the required augmented reality programs can be offloaded to smartphone rapidly.

The goal of offloading is minimizing the processing time and saving energy of wearable device as much as possible. Offloading system provides an environment where program developers only need to annotate the parts to be offloaded. When the program is invoked and the companion mobile device is available, the program will be offloaded to the mobile device automatically. Figure 1 shows the simplified model of our offloading system.

The whole system comprises two parts - one part on wearable devices and the other part on smartphones. On wearable device side, the offloading runtime consists of a proxy and a profiler. The proxy handles control and data transfer for offloaded methods. The profiler instruments the program and collects measurements of the wearable device, such as energy and storage. The profiler keeps detecting network conditions. When the bandwidth falls below a threshold b_0 or it cannot find a mobile device, it will choose to execute the program locally. On the smartphone side, the offloading system provides two similar components, a profiler and a server, which perform similar jobs with their wearable side counterparts. When the battery status of the smartphone falls below a threshold e_0, any offloading task will be refused at the smartphone side. Regarding network connection, both WiFi and Bluetooth can be employed to communicate between wearable device and smartphones. By default we choose WiFi for offloading because of its higher transmission speed. If WiFi is unaccessible, we will turn to Bluetooth.

One significant difference of the adapted offloading system from traditional ones is that it does not include any remote cloud. Computational tasks are only offloaded to the nearby mobile devices. On one hand, the computation capability of smartphones is adequate for AR applications on wearable devices, so offloading to remote cloud cannot largely enhance the performance. On the other hand, in order to minimize delays of real time tasks, the system will require high bandwidth for connection with remote cloud, which is hardly achievable under current situations. However the bandwidth will not be a problem if tasks are offloaded to nearby smartphones instead. Furthermore, privacy issue is another concern. It is common that the face of some person appears in the camera when wearable devices execute AR applications. There is potential danger that the privacy of the person is violated by a third party when the application is offloaded to remote cloud. If it is offloaded to the user's own smartphone, the privacy is well preserved.

In traditional offloading systems, there exists a solver or scheduler which can determine whether the method shall be executed remotely or locally. It aims to find a strategy to best partition the program in order to minimize the energy consumption on the offloader's side. MAUI [2] employs a linear programming model to make decision. When network conditions are not very good, the solver will select only methods which consume lots of computation to offload. The solver is no longer kept in our system for mainly three reasons. Firstly, augmented-reality applications are among the most energy-consuming applications. Being offloaded to mobile devices will certainly gain more than being executed locally. Secondly, the offloader (wearable device) and the offloadee (smartphone) are close to each other, so the network conditions do not pose a problem. Thirdly, the solver itself will consume time to get an optimal method of partitioning the program. Therefore it could lead to longer overhead and inevitably influence the user experience negatively.

Overall we give following guidelines of computational offloading for AR applications on wearable devices:

- Only offload programs to nearby companion mobile devices
- Consider all possible ways of communication, e.g. WiFi and Bluetooth

- Remove traditional scheduler: not using scheduler to decide whether the method should be offloaded
- Create a simplified mechanism at wearable device side to decide whether the method should be offloaded
- Create a simplified mechanism at mobile device side to decide whether to accept the offloaded method
- Offload all common recognition tasks, such as Optical Character Recognition (OCR), object recognition and object detection

3. CASE STUDY

In this section, we introduce three well designed experiments to test the performance of offloading system - AReader, SharpLens and Ubii. All of them are common and practical real-time augmented-reality applications based on Google Glass. The three applications involve computational intensive operations, for instance, image processing and OCR, which are widely used in AR applications.

3.1 SharpLens

SharpLens is an AR application which provides assistance for visually impaired people to access text information around them. It helps users retrieve text information from their view as shown in Figure 2(a). SharpLens contains four different modes: exploring mode, reading mode, meeting mode and driving mode. All detected sections containing text will be highlighted with a rectangle in exploring mode. The key component used in this mode is object recognition, which is the most exhaustive computational part. In reading mode texts will be amplified if they are pointed at by the user, where the key task is gesture recognition. In meeting mode, the application recognizes the text in certain area, in which the key part is OCR. Driving mode is designed for drivers to acquire brief text information of road signs when they are driving. OCR is used in this mode as well. The whole system uses voice command to start the application and switch modalities. For example, a short phrase "OK Glass, start SharpLens" will launch the SharpLens. Text To Speech (TTS) is another frequently-used functionality in SharpLens for speech synthesis. In summary, among all components employed in this application, the most computational intensive tasks are hand gesture detection, text recognition and speech synthesis.

3.2 AReader

AReader is a wearable system to digitally manipulate paper transcripts with Augmented Reality. It can assist users to perform virtual operations on the paper text like text amplification to improve user experience. All contents are shown on the AR overlay, which is over physical surface. Figure 2(b) presents one function mode (Amplification) of AReader. In this application, the main functional modes include text amplification, translation, glossary reference and text clipping. Voice command and finger touching can both be employed to switch between different modes. The text amplification is designed for reading small texts. The translation mode will be called when the finger moves around a word. The glossary reference and corresponding reference will be popped up just beside the finger. In text clipping mode, readers are able to clip, paste and save the contents.

In AReader, the most time-consuming tasks are image preprocessing, hand detection and text recognition. All images taken shall undergo preprocessing like Gaussian blurring to remove noise. Hand detection is one basic operation of the system. In each functional mode, we firstly need to detect the finger correctly in order to properly locate and amplify or translate the texts. Text recognition, namely OCR, is based on a tess-two library [8].

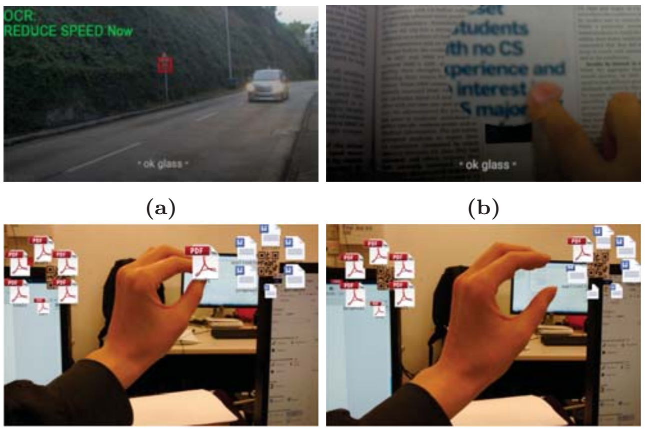

Figure 2: (a) Text recognition with SharpLens (b) Text amplification with AReader (c) File copying between computers with Ubii (d) File pasting between computers with Ubii

3.3 Ubii

Ubii is short for Ubiquitous Interface and Interaction. It is an Graphical User Interface (GUI) where user can interact with physical objects such as computers, project screens and printers. Common operations include document copying, printing and displaying on a projector screen via hand gestures.

In Ubii, we leverage the hand gesture pinching to interact with objects. A series of pinching gestures are defined to represent different operations. Those gestures include picking, dropping, dragging, rotating and zooming. Picking, dropping, dragging are operated on selected items as their name imply. For example, zooming is for scaling up and down the item size and rotating is for file selection in a ring menu. A combination of different gestures can realize a complete function. Picking an image file from the computer, dragging and dropping it to the printer means to print the image. Figure 2(c) and Figure 2(d) illustrate the process of hand pinching and dragging to transmit a file between two computers. Ubii system can be regarded as a finite-state machine, where each operation is a state. Similar with SharpLens and AReader, the most computational intensive task is gesture recognition in Ubii.

4. EXPERIMENT RESULTS

Several experiments were designed to test the performance of offloading on the above three applications. To have a synthesized evaluation, we have conducted experiments on every mode in each application. For each mode the application is tested with and without offloading. The performance is measured in two aspects - energy consumption and execution time. Google Nexus 5 (2.26GHz CPU and 32GB ROM)

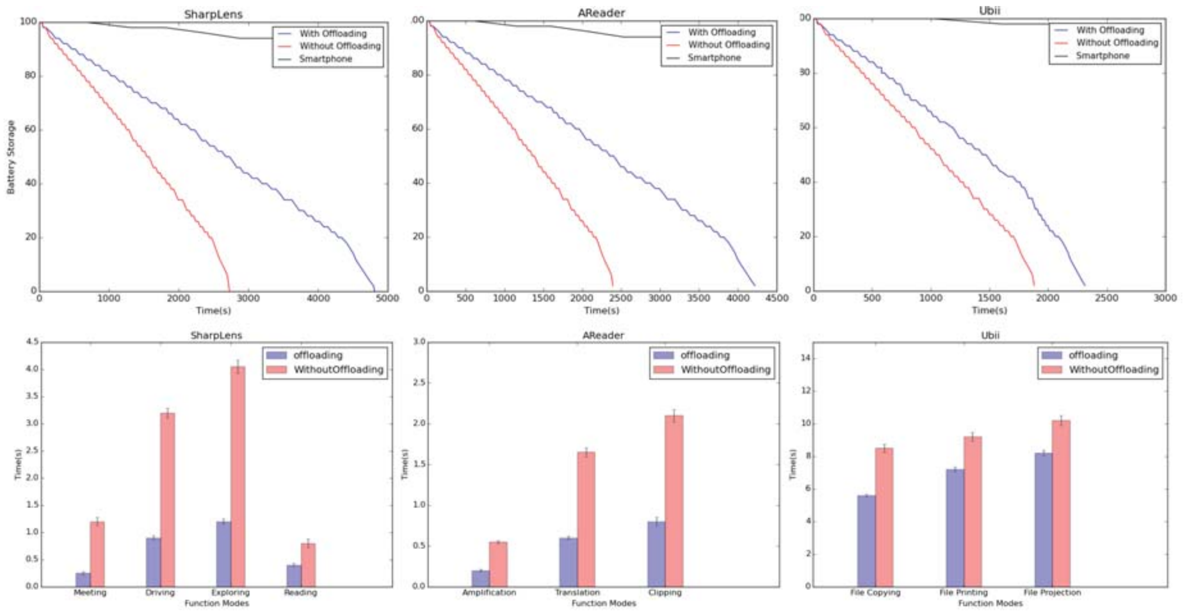

Figure 3: Time cost and energy consumption in different applications

is used as the smartphone to execute offloaded tasks. Data is transmitted via WiFi.

Figure 3 shows the time cost per frame of system and energy consumption in three experiments with and without offloading. In SharpLens and AReader, offloading helps to reduce the computation time to at least 50% in each mode with the battery endurance of Google Glass doubled. On smartphone side, about 6% to 8% of its electricity is consumed in order to do the offloaded computation as whole. The performance improvement by offloading in Ubii is not as prominent as SharpLens and AReader. Computation time is reduced by about 25% and battery life is prolonged by about 30%. The main reason is that there are not so many computational intensive tasks that can be offloaded in Ubii (i.e. gesture recognition) compared with the other two applications (i.e. OCR, finger detection, object recognition).

5. CONCLUSIONS

In this paper we propose several guidelines for designing AR applications on wearable devices and present an improved offloading system based on the guidelines. Our offloading system is lightweight and easy for programmers to offload computation intensive tasks. As we leverage the companion mobile device for offloading, multiple network connection methods can be employed. With experiments conducted on augmented reality applications on Google Glass, our offloading system and guidelines are proved promising. On average, it can achieve energy saving by 20% to 30% and reduce 50% computation time compared with local execution.

Acknowledgements

This work has been partially supported by the European Commission under the Horizon 2020 Program through the RAPID (H2020-ICT-644312) project.

6. REFERENCES

[1] B. G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, pages 301–314. ACM, 2011.

[2] 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 international conference on Mobile systems, applications, and services*, pages 49–62. ACM, 2010.

[3] A. Dou, V. Kalogeraki, D. Gunopulos, T. Mielikainen, and V. H. Tuulos. Misco: a mapreduce framework for mobile systems. In *Proceedings of the 3rd international conference on pervasive technologies related to assistive environments*, page 32. ACM, 2010.

[4] N. Fernando, S. W. Loke, and W. Rahayu. Honeybee: A programming framework for mobile crowd computing. In *Mobile and Ubiquitous Systems: Computing, Networking, and Services*, pages 224–236. Springer, 2013.

[5] M. S. Gordon, D. A. Jamshidi, S. A. Mahlke, Z. M. Mao, and X. Chen. Comet: Code offload by migrating execution transparently. In *OSDI*, pages 93–106, 2012.

[6] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan. Towards wearable cognitive assistance. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pages 68–81. ACM, 2014.

[7] C. Shi, K. Habak, P. Pandurangan, M. Ammar, M. Naik, and E. Zegura. Cosmos: computation offloading as a service for mobile devices. In *Proceedings of the 15th ACM international symposium on Mobile ad hoc networking and computing*, pages 287–296. ACM, 2014.

[8] Theis. R.tess-two. `https://github.com/rmtheis/tess-two`, 2011.