Circa: Offloading Collaboratively in the Same Vicinity with iBeacons

Xueling Lin*, Jingjie Jiang*, Bo Li*, and Baochun Li+

*Department of Computer Science and Engineering, Hong Kong University of Science and Technology
+Department of Electrical and Computer Engineering, University of Toronto

***Abstract*—Code offloading to remote infrastructures has been a common practice for mobile users who seek extra power or computing resources to perform computation-intensive tasks. Existing works, however, have so far mainly focused on code offloading from a single mobile device to remote cloud servers, which restricts the potential of code offloading only to devices with available Internet access. In this paper, we propose *Circa*, a new framework that demonstrates the feasibility of code offloading among multiple mobile devices in close proximity to one another, leveraging the presence of *iBeacons*. Our objective is to eliminate the costs incurred by running virtual machine instances in the cloud, and the need to connect remotely to the cloud as well. With the assistance of iBeacons, devices in the same vicinity can discover and support one another through collaborative code offloading with short-range communication, obviating the need for centralized servers. We have implemented *Circa* on the iOS platform and validated its feasibility using iOS devices. According to our experimental results, with more than two collaborators, *Circa* is capable of reducing the total execution time of an offloaded task substantially, while preserving satisfactory performance of the mobile application.**

I. Introduction

With the dominating prevalence of smartphones, native applications on mobile and wearable devices have blossomed to serve as more convenient substitutes to web applications. Nevertheless, since mobile devices are still resource constrained, those computation-intensive or power-hungry applications are still considered as heavy burdens on these devices. For instance, a smartwatch with an energy-efficient CPU may exceed its computational limits if it were to transcribe long voice messages locally.

To alleviate this problem, the existing literature has resorted to the technique of *code offloading*, in that a portion of a computational task can be offloaded to servers in the cloud, with an abundance of computational resources [1]–[3]. By offloading to the cloud, it is feasible to run much more computation-intensive applications on mobile and wearable devices, such as face recognition, augmented reality and natural language processing. The increased level of parallelism may also improve the performance of these applications. Unfortunately, offloading to cloud computing platforms is not always guaranteed to be time efficient and energy conserved. When the network bandwidth is fairly limited, it may be too slow to transmit data between mobile devices and remote servers; when the network status is highly unstable, maintaining a connection to the cloud may even consume more energy than performing the computation locally. To make matters worse, some devices, such as smartwatches, may not even have access to the cloud, due to their inherent hardware limitations.

In this paper, we propose to detour from the conventional wisdom of offloading to the cloud, and instead focus on the concept of *collaborative offloading* among multiple mobile devices that are in close proximity to one another. With the presence of *iBeacons* [4] that makes a wide range of new location-aware applications possible, devices entering into close proximity to one another can discover each other through the unique identifications distributed by a common iBeacon. A device without sufficient computing capability or battery life can then turn to its "neighbors" for help. We design and implement *Circa*, a new collaborative offloading framework that identifies and selects a group of mobile devices in the same vicinity, and involves them into the same offloading task. With such collaborative offloading in the same vicinity, the costs involved in running instances in the cloud and setting up connections to the cloud will be eliminated, or at least reduced significantly. We have implemented *Circa* on the iOS development platform, and evaluated its effectiveness and performance in a number of practical scenarios. According to our experimental results, with more than two collaborators, the total time spent on the execution of the task offloaded can be reduced substantially.

The remainder of this paper is organized as follows. Sec. II discusses our contributions in the context of related work. In Sec. III, we introduce iBeacons and describe our application scenarios. In Sec. IV, we present the design of collaborative offloading. We evaluate the *Circa* framework through prototype-based experiments in Sec. V. We conclude this paper with discussions of possible extensions in Sec. VI.

II. Related work

Most of existing works, such as *MAUI* [1], *CloneCloud* [2], *ThinkAir* [3] and *COMET* [5], mainly focus on code offloading between mobile devices and cloud servers. With the objective of reducing local energy consumption, such frameworks try to determine which parts of an application could be offloaded to a remote server at runtime. The offloading granularities vary from function calls to running threads. Involving remote servers, however, requires an Internet connection through cellular or Wifi networks. Since the network status between cloud servers and a mobile device is highly unstable, the device may lose its Internet connection to remote servers before the computation finishes. Moreover, some devices, such as smartwatches, may not even have Internet access due to their hardware constraints.

To investigate how mobile devices can function as resource providers and mutually help each other, Doolan *et al.* in [6] design a mobile version of the standard message passing interface over Bluetooth. They employ a fully interconnected mesh structure so that all nodes can directly communicate with each other. Due to the limited communication range of Bluetooth, only devices in a small physical area can form such a fully-interconnected local network. Nevertheless, it is non-trivial to dynamically identify such fully connected clusters under mobile environments. In contrast, *Circa* utilizes the deployment of iBeacons to discover all the devices within close proximity to each other, and enables local cooperation among these devices.

Derived from Hadoop [7], *Hyrax* [8] also explores the possibility of using mobile phones as resource providers. However, it is required to entail a central server to collect device information and coordinate computation jobs. Therefore, Hyrax shares the same deficiencies as the remote offloading frameworks: the unpredictable delay between a remote server and a device may lead to severe performance degradation, incurring intolerable computation latencies. Furthermore, Hyrax only works for a set of smartphones that connect to each other via plain TCP/IP sockets, and thus cannot be applied to a realistic setting where smartphones do not have global, unrestricted IP addresses. Leveraging iBeacons, devices in our framework can efficiently identify each other through the unique IDs distributed by their neighboring beacons.

III. Background and Motivation

In this section, we first introduce iBeacon, a new wireless location-based transmitter that we adopt in *Circa* to detect mobile devices in its proximity. Then we illustrate two practical scenarios that motivate the design of the *Circa* framework.

A. iBeacon: A Proximity Detector

Introduced in iOS 7, iBeacon is an exciting technology proposed by Apple Inc. to enable new location-aware information and services for mobile devices. Leveraging Bluetooth low-energy (BLE) for a short-range communication, a device with iBeacon technology can establish a region around itself. Each iBeacon (called a beacon) broadcasts a 20-byte unique ID, which is divided into three sections: proximity UUID (16 bytes), major number (2 bytes) and minor number (2 bytes). A beacon continuously broadcasts its unique ID via BLE to devices in its close proximity. The broadcast coverage of a beacon could be as small as 2 inches and as large as 230 feet away. Any mobile device entering such coverage of a beacon can receive its unique ID without *a priori* and explicit pairing procedures.

A related event will be triggered by the application of a mobile device when it enters or exists the covered region of the beacon, which can be applied to inform the device

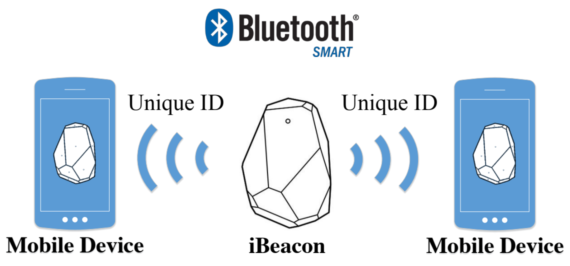

Fig. 1. The way an iBeacon works with mobile devices.

about its geographical location changes [4]. The coverages of different beacons overlap with each other. Mobile devices in the overlapped area can pick up the unique IDs broadcasted by multiple beacons, and estimate their distances to those beacons by measuring the received signal strength (RSSI) [9]. The closer a device is to a beacon, the stronger the corresponding signal it can receive. Thus, the mobile device can obtain its relative location based on the knowledge of its estimated distances to each beacon.

The application scenario of iBeacons is quite simple and straightforward. As shown in Fig. 1, all the devices residing within a beacon's coverage receive the unique ID of the corresponding beacon. By checking the beacon IDs picked up by the others, a device can easily discover those that stay in the same beacon region with it and form a collaborative group with them. As the key to identify such groups, iBeacons play crucial roles in the *Circa* framework.

B. Motivating Examples

To better demonstrate a few highlights of collaborative code offloading in *Circa*, we present two motivating examples where devices can overcome their hardware limitations and the poor network connection performance caused by their mobility, embracing the benefit of code offloading.

Suppose a scholar is attending a meeting held in a conference hall without available WiFi network. Unfortunately, he forgets to bring his smartphone and merely wears a smartwatch, which has no cellular network. With the *Circa* framework supported by iBeacons, the scholar can still enjoy speech recognition and face recognition applications, by sending the speech and image data to nearby smartphones with available resource and battery via Bluetooth, and obtaining the results afterwards. For another example, a group of users are staying in front of a piece of artwork in a museum. Each user uses his mobile device to take a snap of the artwork from his own unique angle. Thanks to the help of iBeacons, they can easily distinguish each other as nearby collaborators, and thus are capable of building the 3D model of the artwork together by allocating the 3D modeling task among each other.

The first example above implies the potential benefit brought by collaborative offloading when hardware limitations or network disconnections are presented in some mobile devices. In contrast, the second example demonstrates the value of collaborative offloading in location-based activities where the collaborating devices share the same objective to complete a computationally intensive task in a parallel fashion.

One essential incentive of collaborative offloading is that all of the participating devices should provide some types of original resource before they enjoy the result. In the first example, the user with a smartwatch shares the audio recordings of a presentation among the nearby users before he can obtain the translation result. Since all the collaborators are geographically close to each other, the probability that they share the same interests is rather high. After collaboration, they can share the text version of the presentation together. As for the second example, everyone donates a snap of the artwork from his own angle before obtaining the complete 3D model. Therefore, in both scenarios, the free rider problem is inherently prevented.

IV. Design

In this section, we present the design of the *Circa* framework. *Circa* consists of a code offloading workflow that involves four basic phases. In the first phase, all mobile devices that stay in the region of a beacon establish connections with one another. In the second phase, these devices are ranked according to their battery levels, memories and mobility patterns. *Circa* then determines the quantity of collaborators in the third phase. Finally, the corresponding offloading proceeds via Bluetooth in the fourth phase.

A. Phase 1: Establish Connections based on iBeacon

In this phase, we aim at setting up connection between each pair of devices that are within the region of a beacon. There are several beacons that broadcast their unique IDs in a certain area. Each mobile device that enters in the Bluetooth transmission regions of the beacons will automatically pick up the ID of each beacon. Once a device starts our application, it can detect all beacons in its proximity and estimate its distance to those beacons. Devices probe the signal of beacons every second.

Afterwards, a peer discovery mode inherited from the GameKit framework [10] is applied to establish connections between devices. Specifically, a GKSession object provides the devices with the ability to discover and connect to nearby mobile devices using Bluetooth. Devices connected to an ad-hoc wireless network are known as peers. A peer is synonymous with a session object running inside our application. Each peer creates a unique peer identification string (session ID), which is used to identify itself to other peers in the network [11]. In the *Circa* framework, sessions discover other peers based on a session mode, which is set when the session is initialized according to the role of the devices as explained below.

A mobile device should choose one of the following three roles:

- *Server:* A mobile device acts as a server of a session when it needs to offload a portion or all of its current task to other devices, if at least one of the following 4 conditions is satisfied:

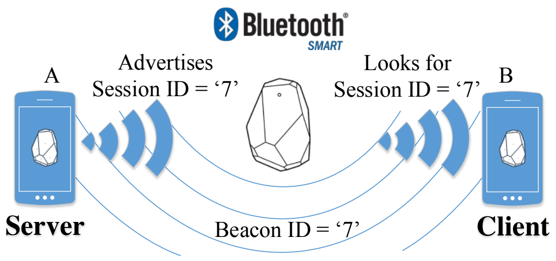

Fig. 2. Peer Connection Example

  a) its remaining battery capacity falls below a certain threshold;
  b) according to its history profiling, both of the energy consumption and time duration of the requested task exceed specific thresholds;
  c) due to hardware constraints, offloading the current task to other devices is of necessity, e.g. the speech recognition task run by the smartwatch mentioned in Sec. III.
  d) it is required to participate in a task that demands collaboration with others, e.g., the collaborative 3D modeling task mentioned in Sec. III;
- *Client:* A device serves as a client if it is willing to accept an offloading request, when both of the following 2 requirements are met:
  a) its remaining battery capacity is over a certain threshold;
  b) according to its history profiling, both of the energy consumption and time duration of the requested task are below specific thresholds, or it is a task that demands collaboration with others.
- *Bystander:* A device acts as a bystander, if it is neither in demand of offloading tasks to others, nor willing to accept offloading requests from others. A bystander does nothing in the whole process.

A server advertises its service type with a session identification string (sessionID) which is the same as the unique ID of the beacon that it picks up. If a server collects more than one beacon ID, it will advertise a set of session IDs that contains every beacon ID it obtained. A client records all the beacon IDs it has picked up as session IDs. It keeps listening for the session IDs broadcasted by other servers and only connects to those servers that have one or more matching session ID. Thus, devices are able to establish connections with one another if and only if they stay in the region of the same beacon.

We illustrate the first phase through an example shown in Fig. 2. Suppose there are two mobile devices, A and B. They stay in the region of a certain beacon, whose unique ID is ‘7’ for short. When A and B start our application, they can detect all beacons in proximity as well as their up-to-date distances. Meanwhile, A and B record the session ID which is exactly the same as the unique ID of the beacon (‘7’). When A needs to offload tasks, it directly advertises '7' as its session ID. If B is willing to collaborate with others, it will start searching for other devices that are broadcasting '7' as the session ID. Then A and B are able to connect to each other successfully as a peer with the session ID '7', and proceed to the next phase.

B. Phase 2: Rank the Potential Collaborators

Although only devices that are geographically close to each other can set up connections, it is still possible to encounter disconnection problems if some collaborators walk away during the offloading process without prompt or just simply shut down their devices. To avoid disconnection in the middle of an offloading process, we need to conduct a reliability analysis to select the most reliable devices as actual collaborators.

There are a few principles that a mobile device should satisfy to be a qualified collaborator. We assume that currently there are m mobile devices connected to a server. After connections are established, each client sends a tiny piece of message to the server, which includes its battery level (in percentage), available memory (in MB) and the time span (in seconds) that it has been stayed in the current region. Based on the reliability analysis, the server then generates a ranked list of the m devices. The following two key observations guide our selection process.

Observation 1: devices with higher battery and available memory should be ranked higher. Since the probability that they run out of battery or memory during the offloading process is much lower, it is less likely for us to encounter disconnection if they are chosen to be the collaborators. Let b_i be the battery level (in percentage) of the i-th mobile device among the $1, 2, ..., m$ devices. Let a_i be the available memory (in MB) of the i-th mobile device. Thus the energy level E_i of the i-th mobile device can be formulated as follows:

$$E_i = b_i * a_i \tag{1}$$

Observation 2: devices that stay in the same region for too short or too long should be ranked lower. Mobile devices that enter in the region for just a few seconds are more likely to be passers-by, who will leave the region in a short time. On the contrary, those who have stayed in one region for a long time also tend to leave soon, since they may finish their tour in the current region soon, and would like to move to another region. Hence the probability for disconnection would be higher if we select them as collaborators.

We use a *normal distribution* to represent the availability of a mobile devices, depending on the time that it has already stayed in the current region. Let μ be the most common time span (in seconds) that those qualified collaborators have been staying in the current region. Let σ be the standard deviation of the time span (in seconds). Let t be the time span (in seconds) that the i-th mobile device has stayed in the current region. Thus the availability $\varphi_i(t)$ of the i-th mobile device can be formulated as follows:

$$\varphi_i(t) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(t-\mu)^2}{2\sigma^2}} \tag{2}$$

Finally, let R_i be the reliability level of the i-th mobile device, which can be formulated as follows:

$$R_i = E_i + \alpha\varphi_i(t) = b_i a_i + \alpha \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(t-\mu)^2}{2\sigma^2}} \tag{3}$$

In this function, α is introduced to combine the energy level and availability of a single device. Therefore, arranging reliability of each device from high to low, a ranked list of these m devices can be generated. Let $Q(m_1, m_2, m_3, ...)$ be the ranked list obtained. A server will choose those devices with higher rankings as its collaborators.

In the availability formulation of the *Circa* framework, μ is set to be 20s, σ is set to be 10s and α is set to be 500.

C. Phase 3: Determine the Quantity of the Collaborators

In this phase, we need to determine the quantity of the mobile devices we need to involve. In order to achieve the best performance with least energy consumption and transmission latency, it is important for us to decide that how many devices should be picked out as collaborators from the ranked list, which is obtained in the second phase.

The main challenge of this problem is to balance the trade-off among energy cost, transmit latency and application performance. More collaborators lead to better application performance for the following reasons. First of all, after offloading part of its computation task to other devices, a single device incurs less energy cost. Moreover, since the collaborators share a common object to achieve a positive outcome, the performance can be enhanced if the execution sequence of the application can be reordered for increasing the level of parallelism. The total time required to finish the whole task could also be cut down. However, involving more nearby devices as potential collaborators will inevitably cause longer delay, since it takes extra time for all the devices to connect to each other.

In the current implementation of *Circa*, a server chooses all the devices whose reliability level are higher than a certain threshold as its collaborators. The threshold is set to be 15.00.

D. Phase 4: Offload the Task

After a group of collaborators have been chosen, the offloading of tasks starts immediately. The whole offloading process is conducted via Bluetooth.

There are two working schemes in *Circa*. In the first scheme, a single server device broadcasts its offloading request to other mobile devices that are within the same region, e.g. when processing the speech recognition task run by the smartwatch mentioned in Sec. III. The offloaded task is then divided into multiple smaller subtasks, each of which is transmitted to one client device. The workload of each subtask is similar to each other to simplify the task allocation. In this case, there are one server and multiple clients. A mobile device acts as either a server or a client. In the second scheme, all the mobile devices share the same goal to complete one computation-intensive task. Each of the devices provides its own original resource and takes its own responsibility to accomplish the given part of the computation, e.g., when processing the collaborative 3D modeling task mentioned in Sec. III. In this case, there are multiple servers and multiple clients. A mobile device serves as both a server and a client at the same time.

Last but not least, in *Circa*, under the situation that one of the users leaves the region and disconnects with others, the intermittent results are returned to another device for further processing.

V. Performance Evaluation

In this section, we evaluate the performance of *Circa* to validate its feasibility and efficacy. We first describe the hardware configurations, test applications and performance metrics used in our experiments. We then present and analyze the experimental results under different scenarios in detail.

A. *Experiment Setup*

Our prototype consists of three smart devices: an iPhone 5, an iPhone 4s and an iPad 3. The detailed information of the configurations for the tested devices are listed in Table I.

TABLE I
Configurations of mobile devices tested

Devices	iPhone 5	iPhone 4s	iPad 3
Version	iOS 7.1.2	iOS 7.1.2	iOS 7.1.2
Capacity	27.9GB	28.3GB	27.8GB
Avaliable	9.6GB	21.8GB	4.2GB
Battery	15% ∼ 20%	45% ∼ 55%	90% ∼ 100%

We deploy 3 beacons bought from Estimote Inc. in our experiment environment, acting as the iBeacon transmitters in *Circa*. Each beacon has a square chip inside, which is s 32-bit ARM Cortex M0 CPU with 256KB flash with a 2.4GHz Bluetooth low-energy radio [9]. During the experiments, all of the beacons are attached to the plain wall in order to avoid possible signal distortion.

To test the practical performance of *Circa*, we develop a speech recognition application, which records a wav file of the speaker, then transmits the file to Google Speech API to obtain the corresponding text result. The application divides the wav file into several smaller files according to the quantity of selected collaborators.

We examine *Circa* under different scenarios in our experiment. In the first scenario "single device", the original device carries out the speech recognition task by itself. In the second scenario "2 collaborators" (1 server and 1 client), the original device switches to the airplane mode and only turns on the Bluetooth for connection, which means that its WiFi and cellular network are both cut off. This original device then acts as a server. There is another device that is willing to act as a client to accept the task from the server. In other words, there are 2 collaborators, one serves as a server and the other serves as a client. In the third scenario "3 collaborators" (1 server and 2 clients), the original device still serves as a server, while there are two other devices acting as clients to accept the task offloaded from the server. The wav file from the server will be divided into two pieces of files with identical amount

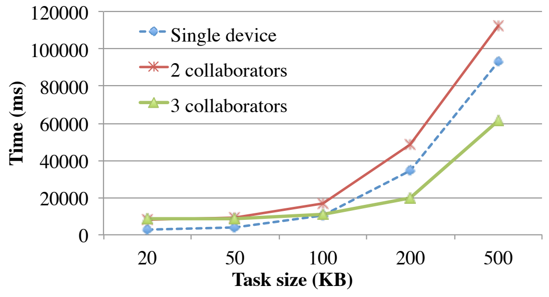

Fig. 3. The total execution time with different task sizes. Distance = 1m. Speed = 0m/s. No obstacle.

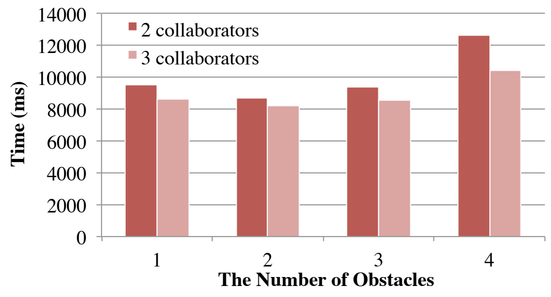

Fig. 4. The total execution time in different environment. Task size = 50KB. Distance = 1m. Speed = 0m/s.

of data, which will be offloaded to the 2 clients for speech recognition. In this situation, there are 3 collaborators in total.

In all the experiments, the iPhone 5 acts as the only server, while iPhone 4s and iPad 3 act as potential clients.

We measure the time duration of a whole task as an indicator of its performance, since the speech recognition task is time-sensitive. All of our measurements are performed under stable network conditions, with all the mobile devices running in standalone environments, in which all other applications and background tasks are shut off, with the screen on.

B. *Performance Analysis*

The first experiment is conducted to compare the total execution time of the whole task with single device, 2 collaborators (1 server and 1 client), and 3 collaborators (1 server and 2 clients). As shown in Fig. 3, it takes longer time to complete larger tasks. Compared to the original scheme with only one device, if there are 2 collaborators, there will be additional connection time, which is 5000ms on average. However, if more than two devices are involved in the task, the total execution time for the same task will be shorter than the local execution. The benefit of offloading becomes more significant when the task is larger. The main reason for this phenomenon is the degree of parallelism. With 3 collaborators, we can divide the task into two smaller pieces so that they can be processed at the same time, while the transmission delay is also around 5000ms, remaining the same with that of 2 collaborators.

In the second experiment, whose results are shown in Fig. 4,

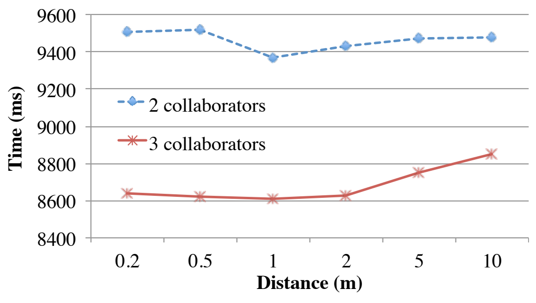

Fig. 5. The total execution time with varying distances between the devices. Task size = 50KB. Speed = 0m/s. No obstacle.

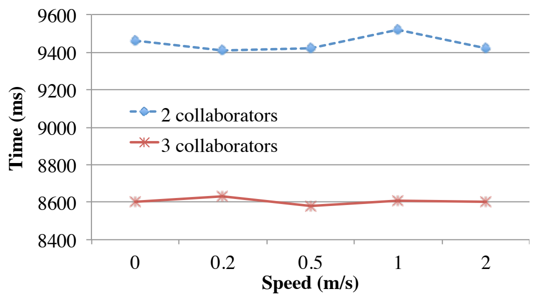

Fig. 6. The total execution time with varying speeds of the devices. Task size = 50KB. Distance = 1m. No obstacle.

we measure the execution time of the task under different environments with obstacles. The reason for conducting such experiments is that Bluetooth signal could be easily affected by physical surroundings due to distraction or absorption. We use 1, 2, 3 and 4 cardboards as obstacles to block the signal from the beacon in different directions. The devices stand in a line, pointing in the same direction to the beacon. According to the result, only when we use 4 cardboards to block all 4 directions around the beacon, will the total execution time increase significantly due to minor connection latency.

The third experiment examines the impact of the distances between devices during task execution. The distances shown in Fig. 5 refer to the vertical distances between every two devices. Based on the experimental results, we can conclude that the distances between the devices do not affect the offloading performance if the devices all stay in the broadcasting region of the same beacon. However, there will be additional connection latencies if the devices are too close to each other (less than 1m). The fourth experiment compares the total execution time with varying moving speeds of the devices during the offloading process. According to experimental data shown in Fig. 6, the speeds of the devices do not affect the process time largely, as long as the devices stay in the Bluetooth transmission region of each other.

VI. Concluding remarks and future work

In this paper, we have designed and implemented *Circa*, a collaborative code offloading framework, leveraging the presence of iBeacon. Unlike the involvement of cloud servers in existing schemes, *Circa* is completely localized and requires no Internet connection. Devices within a local area can discover and help each other with the assistance of iBeacons as long as their Bluetooth functions work properly. As a proof of concept, we have implemented a prototype to evaluate the feasibility and performance of *Circa*. Experimental results with realistic settings have shown that code offloading incurs negligible (if any) delay with 2 collaborators. Fortunately, with more than 2 collaborators, the performance will be improved since total execution time is reduced, when compared to local execution of the same code.

As for further discussion, in order to improve the performance of the *Circa* framework, it is possible to involve a cloud server which connects to each device via WiFi or cellular network. The cloud server will be able to monitor the devices more efficiently and record the beacons in their vicinity. With a cloud server, the initiator devices (servers) no longer need to spend extra battery or memory on the reliability analysis of their potential collaborators (clients). However, the benefit might offset the latency and energy consumption caused by long-range network connection to cloud.

Another possible extension to *Circa* lies in the division of an offloaded task. Currently, the task is divided evenly according to the quantity of collaborators. Nevertheless, it would be more reasonable if the size of each divided part depends on the battery level and availability state of its targeted device. We leave a more sophisticated division algorithm for future work. Moreover, We will consider involving a realistic mobility model in our mobility management in order to improve the reliability of the framework. We will also complete our experimental evaluation of the prototype system by involving more devices.

References

[1] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proc. ACM MobiSys*, 2010, pp. 49–62.

[2] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proc. ACM EuroSys*, 2011, pp. 301–314.

[3] 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. IEEE INFOCOM*, 2012, pp. 945–953.

[4] "iBeacons for developers," https://developer.apple.com/ibeacon/.

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

[6] D. C. Doolan, S. Tabirca, and L. T. Yang, "MMPI: a message passing interface for the mobile environment," in *Proc. ACM Conference on Advances in Mobile Computing and Multimedia*, 2008, pp. 317–321.

[7] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Proc. IEEE Symposium on Mass Storage Systems and Technologies (MSST)*, 2010, pp. 1–10.

[8] E. E. Marinelli, "Hyrax: cloud computing on mobile devices using mapreduce," DTIC Document, Tech. Rep., 2009.

[9] "Estimote Beacon API," http://estimote.com/api/.

[10] A. Horovitz, K. Kim, J. LaMarche, and D. Mark, "Peer-to-peer over bluetooth using game kit," in *More iOS6 Development*. Springer, 2013, pp. 251–293.

[11] "Game Kit Programming Guide," http://nathanmock.com/files/com.apple.adc.documentation.AppleiOS6.0.iOSLibrary.docset/Contents/Resources/Documents/#documentation/NetworkingInternet/Conceptual/GameKit_Guide/GameKitConcepts/GameKitConcepts.html.