Opportunistic Mobile Crowd Computing: Task-dependency Based Work-Stealing

Sanjay Segu Nagesh, Niroshinie Fernando, Seng W. Loke, and AzadehGhari Neiat
ssegunagesh,niroshinie.fernando,seng.loke,azadeh.gharineiat@deakin.edu.au
School of Information Technology, Deakin University
Geelong, Australia

Pubudu N. Pathirana
pubudu.pathirana@deakin.edu.au
School of Engineering, Deakin University
Geelong, Australia

ABSTRACT

Mobile devices are ubiquitous, heterogeneous and resource constrained. Execution of complex tasks in mobile devices are resource demanding and time-consuming, forcing developers to offload portions of the complex task to cloud or edge computing resources. Task offloading becomes increasingly challenging due to intermittent Internet connectivity, remote resource unavailability, high costs, latency, and limited energy of the mobile device. A mobile device user is typically surrounded by other mobile devices, which can be leveraged to collaboratively compute a resource-intensive task. With the help of a work sharing framework, it is feasible for devices to communicate and collaborate. However, some mobile devices are incapable of computing complex portions of the task, and some can compute in accelerated mode. In this demonstration, we introduce Honeybee-T a collaborative mobile crowd computing framework that uses a work-stealing algorithm. The algorithm allows work sharing with collaborating devices based on devices' computational ability and task-dependencies. The experiments show that by employing Honeybee-T framework, when compared to monolithic execution of a large compute-intensive task, there is a considerable performance gain, as well as energy savings.

1 INTRODUCTION

Mobile applications have historically been developed considering the end-user mobile devices to be limited in resources, especially for sophisticated applications like video editing, and data analytics. To overcome the inherent resource constraints of mobile computing, application developers began to integrate cloud or edge resources to support complex computations [9, 11, 16]. While *Cloud computing* (CC) enables on-demand access to computing resources [3], there are issues with latency and bandwidth, especially in circumstances with intermittent connectivity. The concept of *Mobile edge computing* (MEC), which places additional computing servers between the remote cloud and the end-user devices, has been introduced as a solution [10, 15]. In MEC, an edge server is primarily responsible for resource sharing and communicating with remote resources for task offloading [8]. In hazardous environments, with node mobility, edge resource unavailability or issues with network infrastructure, it can be challenging for the offloaded tasks to be completed in a way that adequately fulfils Quality of Service (QoS) requirements [1, 5].

As an alternative solution, to complement the existing CC and MEC paradigms, the increasing number of mobile devices surrounding a user, prompts an opportunity to share each others' distributed computational resources without the need for continuous connectivity or supervision from centralised infrastructure. This concept of a local mobile device resource cloud, where collections of mobile devices and their users collaboratively harness the computational resources of those mobile devices, is called a 'Mobile Crowd Computing' (MCdC) [2, 6]. Honeybee [6, 7], Oregano [12, 13], and Serendipity [14] are some examples of MCdC frameworks. The Honeybee [6, 7] framework demonstrates the feasibility of MCdC using Bluetooth and WiFi-Direct technologies, and employs a work-stealing algorithm to facilitate task scheduling [4].

This work builds on previous work of Honeybee, creating a novel extension called Honeybee-T, which handles task dependencies among jobs subject to work-stealing, a complexity which has not been considered for mobile crowd computing in previous works. Dependencies among tasks, where a certain task's processing depends on the results of another task, can be expected in many real-world use cases. If not adequately managed, such dependencies can produce bottlenecks by causing several tasks to be unscheduled until the processing data is available.

This demonstration shows the working prototype of Honeybee-T, facilitating mobile devices of varied capabilities to perform a collaborative computation, with task dependencies. A Human Activity Recognition (HAR) task is implemented as an Android app, using the Honeybee-T framework, and demonstrates performance gains as well as energy savings in a mobile crowd setting.

2 SYSTEM DESIGN

To form a successful collaboration mobile crowd, the system involves at least two mobile devices. A delegator device, where exists a compute-intensive task and one or more worker devices, willing to contribute computational resources and execute parts of the compute-intensive task. This is made possible by breaking the task in to smaller 'jobs' on the delegator, which are added to a 'job pool', and enabling the worker/s to 'steal' chunks of jobs from the delegator. The delegator also keeps executing jobs from the job pool simultaneously. Both delegator and worker/s keep returning the results as they complete and keep stealing more jobs. There are two main components of the system: (1) Job Pool and (2) Handshake.

ACM MobiCom '22, October 17–21, 2022, Sydney, NSW, Australia

ACM ISBN 978-1-4503-9181-8/22/10…$15.00
https://doi.org/10.1145/3495243.3558751

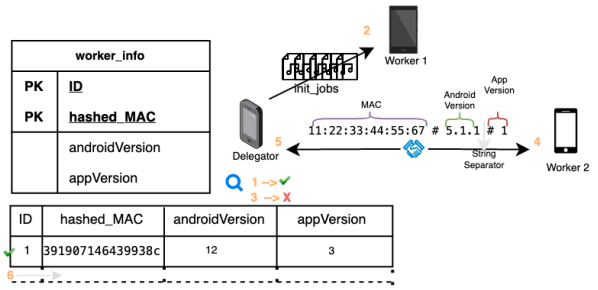

Figure 1: Honeybee-T Handshake Scenarios

The **JobPool** holds a collection of jobs at the Delegator. It is implemented using a double-ended list, and the jobs are queued and dequeued from either end of the list based on the steal requester. When a steal is made the job is removed from the queue and information about the stolen jobs are maintained, and the stealing of jobs is done by both delegator and worker devices. We have modified the Job object and made changes to Honeybee's work-stealing algorithm so that sequential dependency-based task stealing is achieved. In the Job object, the *Job* data structure includes 'stage' property to indicate the current stage of data processing. This will then be used as the steal condition, which means that if a device is incapable of performing a *stage i* job, then that job will not be allowed to steal, instead a more capable job will be released.

The **Handshake**: Helps to regulate task-stealing strategy based on the devices' capability. The participating devices' metadata must be collected before the commencement of a collaboration session. The devices' metadata are stored in a SQLLite database, which is used to look-up device information, before deciding whether to initiate a Handshake or not. An overview of the handshake mechanism is presented in Figure 1. The figure represents the following: When the delegator device establishes a socket connection with worker 1, in step 1 the delegator will check its database (DB) worker info table to see if there is a record of the worker 1 app and the Android version. Since the record existed, the delegator did not initiate the handshake, instead in step 2 shared the initial chunk of jobs. Regarding worker 2 post-successful socket connection with the delegator, in step 3 delegators could not find an entry in the table worker info before sharing the initial chunk of jobs. In step 4, delegator initiates a handshake request, and worker 2 will respond to the handshake by sharing its metadata, which is step 5. To avoid future handshakes, at step 6, the delegator will update the worker info table.

3 DEMONSTRATION

This demonstration emulates a scenario where a mobile application requires external resources to provide accurate and timely results, while in an environment with mobility, and limited infrastructure connectivity. Let us take the case of a passenger (John) on a train, travelling with friends, requiring continuous monitoring of body movements due to a health condition. John is keen to utilise the collective 'mobile crowd' of his friends' mobile devices' to obtain timely results while also conserving his phone battery, than using CC and/or MEC services. Emulating this scenario, the demonstration uses a dataset comprised of accelerometer, gyroscope, and magnetometer data, that requires stages of processing and a Machine Learning (ML) model for activity recognition, requiring capable processors. Figure 2 presents the conventional execution flow of the HAR task. The first stage is the data preprocessing stage; the purpose of the data preprocessing stage is to remove any noisy signals recorded along with the activity data. In our experiments, 'Butterworth LowPass Filter' employed to perform filtering. The second stage is feature extraction, where 'Fast Fourier Transform' over the preprocessed data. In the final stage, using 'TensorFlowLite' library an LSTM model is implemented to perform recognition activity. These three tasks must be performed in a sequential order, hence the dependency.

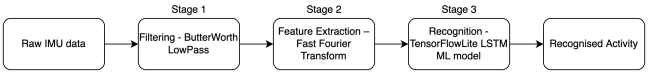

Figure 2: Overview of execution steps involved in HAR

Honeybee-T currently supports three Android versions. 1) Version '1' which does not have *stage* 3 logic and is suitable for devices with Android version 6 and lower. 2) Version '2' has all *stages* and is suitable for devices with Android version between 7 and 11. Finally, 3) Version '3' also has all *stages* and is suitable for devices with Android version 12 and greater. The devices listed in Table 1

Table 1: Devices used in experiments and their roles

ID	Device	App Version	Android Version	Experiment Role
D1	OnePlus 5T	2	10	Delegator / Worker
D2	Pixel 6 Pro	3	12	Worker
D3	Pixel 4	2	11	Delegator / Worker
D4	Pixel 3	2	11	Worker
D5	Moto G	1	5.1.1	Worker

device roles are different in Android versions, as well as in terms of computability. In addition to Google Pixel 6 Pro and Pixel 3 devices, all other devices qualify for the Honeybee-T app with version 2. Google Pixel 6 Pro qualifies for App version 3 and the Moto G device qualifies for App version 1. In all experiments other than monolithic execution, the Pixel 4 or OnePlus5T device is the *delegator* and other devices are *workers*. The figure 3 shows performance and energy gains as a result of collaboratively computing the task. The demonstration will also show a case where the system handles collaborating devices randomly leaving and joining, as can be expected in the emulated train scenario.

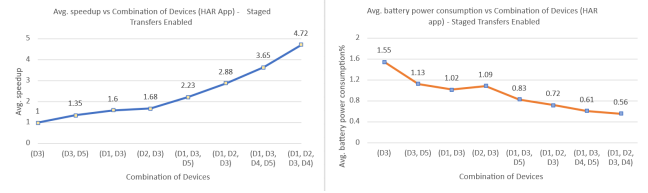

Figure 3: Performance results of average speedup & power consumption vs. combination of devices for HAR application

REFERENCES

[1] 2019. Copyright. In *Digital Twin Driven Smart Manufacturing*, Fei Tao, Meng Zhang, and A.Y.C. Nee (Eds.). Academic Press, iv. https://doi.org/10.1016/B978-0-12-817630-6.00014-X

[2] Kenneth Li Minn Ang, Jasmine Kah Phooi Seng, and Ericmoore Ngharamike. 2022. Towards Crowdsourcing Internet of Things (Crowd-IoT): Architectures, Security and Applications. *Future Internet* 14, 2 (2022), 49.

[3] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. 2010. A view of cloud computing. *Commun. ACM* 53, 4 (2010), 50–58.

[4] Robert D Blumofe and Charles E Leiserson. 1999. Scheduling multithreaded computations by work stealing. *Journal of the ACM (JACM)* 46, 5 (1999), 720–748.

[5] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. 2012. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. 13–16.

[6] Niroshinie Fernando, Seng W Loke, and Wenny Rahayu. 2012. Honeybee: A programming framework for mobile crowd computing. In *International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services*. Springer, 224–236.

[7] Niroshinie Fernando, Seng W Loke, and Wenny Rahayu. 2016. Computing with nearby mobile devices: a work sharing algorithm for mobile edge-clouds. *IEEE Transactions on Cloud Computing* 7, 2 (2016), 329–343.

[8] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. 2015. Mobile edge computing—A key technology towards 5G. *ETSI white paper* 11, 11 (2015), 1–16.

[9] Heyoung Lee, Heejune Ahn, Trung Giang Nguyen, Sam-Wook Choi, and Dae Jin Kim. 2017. Comparing the self-report and measured smartphone usage of college students: a pilot study. *Psychiatry investigation* 14, 2 (2017), 198.

[10] Fangming Liu, Peng Shu, Hai Jin, Linjie Ding, Jie Yu, Di Niu, and Bo Li. 2013. Gearing resource-poor mobile devices with powerful clouds: architectures, challenges, and applications. *IEEE Wireless communications* 20, 3 (2013), 14–22.

[11] Christian Montag, Konrad Błaszkiewicz, Rayna Sariyska, Bernd Lachmann, Ionut Andone, Boris Trendafilov, Mark Eibes, and Alexander Markowetz. 2015. Smartphone usage in the 21st century: who is active on WhatsApp? *BMC research notes* 8, 1 (2015), 1–6.

[12] Pedro Sanches, João A Silva, António Teófilo, and Hervé Paulino. 2020. Data-Centric Distributed Computing on Networks of Mobile Devices. In *European Conference on Parallel Processing*. Springer, 296–311.

[13] Pedro Miguel Castanheira Sanches. 2017. *Distributed computing in a cloud of mobile phones*. Ph.D. Dissertation.

[14] Cong Shi, Vasileios Lakafosis, Mostafa H Ammar, and Ellen W Zegura. 2012. Serendipity: Enabling remote computing among intermittently connected mobile devices. In *Proceedings of the thirteenth ACM international symposium on Mobile Ad Hoc Networking and Computing*. 145–154.

[15] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge computing: Vision and challenges. *IEEE internet of things journal* 3, 5 (2016), 637–646.

[16] Thomas DW Wilcockson, David A Ellis, and Heather Shaw. 2018. Determining typical smartphone usage: What data do we need? *Cyberpsychology, Behavior, and Social Networking* 21, 6 (2018), 395–398.