This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2560163, IEEE Transactions on Cloud Computing

Computing with Nearby Mobile Devices: a Work Sharing Algorithm for Mobile Edge-Clouds

Niroshinie Fernando, Seng W. Loke, and Wenny Rahayu

Abstract—As mobile devices evolve to be powerful and pervasive computing tools, their usage also continues to increase rapidly. However, mobile device users frequently experience problems when running intensive applications on the device itself, or offloading to remote clouds, due to resource shortage and connectivity issues. Ironically, most users' environments are saturated with devices with significant computational resources. This paper argues that nearby mobile devices can efficiently be utilised as a crowd-powered resource cloud to complement the remote clouds. Node heterogeneity, unknown worker capability, and dynamism are identified as essential challenges to be addressed when scheduling work among nearby mobile devices. We present a work-sharing model, called *Honeybee*, using an adaptation of the well-known work stealing method to load balance independent jobs among heterogeneous mobile nodes, able to accommodate nodes randomly leaving and joining the system. The overall strategy of Honeybee is to focus on short-term goals, taking advantage of opportunities as they arise, based on the concepts of proactive workers and opportunistic delegator. We evaluate our model using a prototype framework built using Android and implement two applications. We report speedups of up to 4 with seven devices and energy savings up to 71% with eight devices.

Index Terms—mobile edge-clouds, crowdsourcing, mobile crowd computing, offloading

1 Introduction

Todays environments are becoming embedded with mobile devices with augmented capabilities, equipped with various sensors, wireless connectivity as well as limited computational resources. Whether we are on the move, on a train, or at an airport, in a shopping centre or on a bus, a plethora of mobile devices surround us every day [47], thus creating a resource-saturated ecosystem of machine and human intelligence. However, beyond some traditional web-based applications, current technology does not facilitate exploiting this resource rich space of machine and human resources. Collaboration among such smart mobile devices can pave the way for greater computing opportunities [54], not just by by creating crowd-sourced computing opportunities [29] needing a human element, but also by solving the resource limitation problem inherent to mobile devices. While there are research projects in areas such as mobile grid computing where mobile work sharing is centrally coordinated by a remote server (HTC power to give[1]) and crowd-powered systems using mobile devices (Kamino[2], Parko[3]) a gap exists for supporting collective resource sharing without relying on a remote entity for connectivity and coordination. However such *mobile crowds* (also referred to as mobile edge-clouds [20]) are not meant to replace the remote cloud computing model, but to complement it as given below:

- As an alternative resource cloud in environments where connectivity to remote clouds is minimal.
- To decrease the strain on the network.
- To utilise machine resources of idle mobile devices [55].
- To exploit mobile devices' sensor capabilities which has enabled the *mobile crowdsensing* paradigm [27]. A resource cloud capable of such multi-modality sensing can enable innovative applications.
- As mobile devices are usually accompanied by users, they also possess an element of human intelligence [27] which can be leveraged to solve issues that require human intervention, such as qualitative classification.

A *mobile crowd* can be viewed as a specialized form of a *mobile cloud* which, in turn, can be viewed from two main perspectives:

- migrating the computation and storage in mobile devices to resource-rich centralized remote servers, and
- leveraging the computational capabilities of the mobile devices by having them as resource nodes, as been adopted in research such as the Mobile Device Cloud [21], [46], Hyrax [42], Mobile Edge-Clouds [20], [2], [28], [6], MClouds [45], MMPI [19], Virtual cloud computing for mobile devices [31], and in [55].

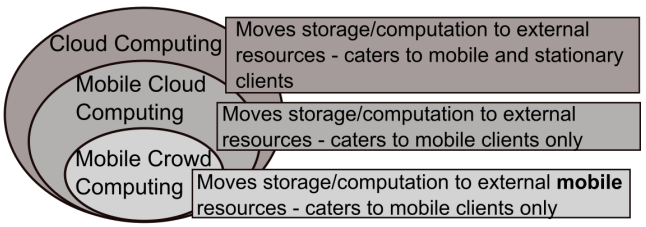

Fig. 1: Classifications of Cloud Computing subsets

Both of these views have the same objective of moving computation and/or storage away from the resource-

This work was supported in part by the La Trobe University Postgraduate Writing-up Award

N. Fernando was with La Trobe University, Australia and now works at Swinburne University of Technology, Australia (e-mail: niro.ucsc@gmail.com)
S.W. Loke and W. Rahayu are with La Trobe University.

1. http://www.htc.com/us/go/power-to-give/
2. http://www.gokamino.com/
3. http://www.parko.co.il/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2560163, IEEE Transactions on Cloud Computing

constrained mobile device to an external entity. As illustrated in Figure 1, the difference lies in the nature of *external resource providers* used to augment the computing potential of mobile devices. The focus of this paper is on mobile crowd (or edge-cloud). In our view, the *human user* of a mobile device is also a resource, which adds an element of *crowd computing* [48] to the mobile cloud as well. Therefore, we refer to this specialized mobile cloud as the *Mobile Crowd*.

There are several unique features that differentiate mobile crowd environments from a typical grid/distributed computing cluster, such as less computation power and limited energy on nodes, node mobility resulting in frequent disconnections, and node heterogeneity [22]. Hence, solutions from grid/distributed computing cannot be used as they are, and need to be adapted to suit the requirements of mobile crowd environments.

This paper presents the Honeybee model, that supports P2P work sharing among dynamic mobile nodes. As proof of concept we present the Honeybee API, a programming framework for developing mobile crowd computing applications. We build on previous work where we initially investigated static job farming among a heterogeneous group of mobile devices in [25], which was followed by a more self adaptive approach in [22] using the 'work stealing' method [8], and in [23] where three different mobile crowdsourcing applications were implemented and evaluated. The progress of our research on work sharing for mobile edge-clouds is illustrated in Table 1.

TABLE 1: Evolution of the Honeybee model for computing with nearby mobile devices

Phase I	**Phase II**	**Phase III**
Simple work farming on Bluetooth [25] • connect to workers via Bluetooth • distribute jobs equally • no load-balancing	Work stealing on Bluetooth [22], [23] • connect to workers via Bluetooth • distribute jobs equally • load-balancing via work stealing after initial job distribution	Enhanced work stealing on Wi-Fi Direct: **current paper** • connect to workers via Wi-Fi Direct • work stealing commences without initial equal job distribution • fault-tolerance methods • periodic resource discovery

In this phase, we have improved the work stealing algorithm of phase II to address the bottlenecks in the transmission of large job data by optimising the job distribution strategy and using Wi-Fi Direct. Phase III is also able to handle random disconnections and opportunistic connections. Beyond our previous work, the main contributions of this paper are, an enhanced stealing method, evaluation of system behaviour, and mathematical bounds for performance. We show considerable amounts of performance gain and energy savings using our system. Although we recognize that incentives, security and trust mechanisms are essential for a successful mobile crowd, these issues are not addressed in this work. For the purposes of this paper, we have assumed that; incentive mechanisms are already in place, and Honeybee is run on a secure environment.

2 Related Work

Offloading computation and storage from mobile devices to an external set of resources, has been explored in the literature [24], [17], [37], [54]. With regards to the resource offloading, current research can be viewed from three main perspectives: offloading to a remote resource cloud[34], [15], [14], [13], [35], [30], to a local cloudlet or local infrastructure[57], [6], and to other mobile devices [23], [42], [18], [31], [21], [45], [28]. Each of the three methods have advantages depending mainly on the existence of high connectivity, additional infrastructure or node encounters respectively. In our work, we focus on the third method, ie., opportunistically sharing work with the surrounding mobile devices, owing to issues with the other two approaches in cases of low network availability and lack of established infrastructure. Furthermore, in Honeybee, we also recognize the potential of using mobile devices as agents of *crowdsourcing*[29], thereby exploiting the collective power of human expertise and machine resources.

In much research regarding mobile work sharing, the existence of a central server has been essential to either co-ordinate jobs among the mobile devices [42],[36], or to offload the work on to [12], [14], [34]. However, our system follows a decentralized job sharing method, with the job scheduling depending entirely on the availability of the participating nodes. The concept of mobile devices forming resource clouds has been discussed by Miluzzo et al. in [45], which identifies key areas of *'MCloud Management'* including periodic resource discovery, formation, fault tolerance, and handling mobility. In Honeybee, we also recognize the need to address the aforementioned areas, plus load balancing, and provide a complete implementation that supports them. An emulation testbed to evaluate the time and energy savings of offloading to a *Mobile Device Cloud* has been implemented in [21]. Such a testbed can be useful for mobile application development using an API such as Honeybee and some of the results reported from their testbed are comparable with our figures. However, our experimental data also suggest that there are additional factors that affect the overall performance such as accommodating random disconnections, unknown node capabilities, and unequal job distributions. Phoenix [51] proposes a distributed storage service using mobile devices in the vicinity, and shows the possibility to ensure data longevity despite autonomous node mobility. Honeybee, on the other hand, focuses on offering computation services rather than storage. In most mobile task sharing systems, Wi-Fi or 3G has been the most used communication protocols, except in the cases such as the MMPI framework [18], which is a mobile version of the standard MPI over Bluetooth, and uses Bluetooth exclusively for transmission, and Cuckoo[34], based on the Ibis communication middleware [62], to offload to a remote resource, and supports Bluetooth with Wi-Fi and cellular. Although Honeybee has used Bluetooth in previous versions, the current implementation uses Wi-Fi Direct due to better speeds and range. FemtoCloud [28] proposes an opportunistic mobile edge-cloud platform that offloads jobs to nearby mobiles, similarly to Honeybee. However, whereas Honeybee does not require prior information about the computational capabilities of the worker nodes to load-balance the task, FemtoCloud's scheduling strategy depends on periodic capability estimations of each worker node.

At the other end of the spectrum, crowd computing[48], [47], [52] has been shown to have the potential to use mobile

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2560163, IEEE Transactions on Cloud Computing

devices in a social context to perform large scale distributed computations, via a static farming method. However, our results show that the work stealing method can provide better results. Social aware task farming has been proposed as an improvement on simple task farming, and social aware algorithms show better performance in their simulation based on real world human encounter traces [48]. In the future we hope to build on this result (social aware task sharing) as an incentive for participation. In [26], human expertise is used to answer queries that prove too complicated for search engines and database systems, and in CrowdSearch [64], image search on mobile devices is performed with human validation via Amazon Mechanical Turk. A generic spatial crowdsourcing platform using smartphones is discussed in [11], where queries are based on location information. Mobile phones are used to collect sensor data on Medusa [53], according to sensing tasks specified by users. In Rankr [41], an online mobile service is used to ask users to rank ideas and photos. These are primarily concerned with the *crowdsourcing* aspect, using mobile devices as tools to access an online crowdsourcing service that is hosted on a remote server. In contrast, Honeybee defines the *crowd* as the surrounding mobile devices and their users, and focuses on sharing the tasks on a *crowd* of local mobile devices with performance gain and saving energy as the main goal. Indeed, results from the above research show us that user participation is at a considerable level, and using micro payments for such 'micro tasks' is viable.

Our work is different from these in terms of using only local mobile resources opportunistically, satisfying the requirements of a mobile device cloud of being *proactive*, *opportunistic* and *load-balanced* while showing speedups and energy savings in an actual implementation. Our focus is on a model that can be used to implement a variety of tasks, not limited to query processing, sensing, or human validation. We compare and contrast features of Honeybee with similar work focused on distributed mobile computation in Table 2.

3 Model and Algorithms

We define *Mobile Crowd Computing* as a group of *dynamically* connected mobile devices and their users using their *combined machine and human intelligence* to execute a task in a *distributed* manner. Such a mobile crowd is comprised of *heterogeneous* devices and could be unknown to each other a priori. Participating mobile nodes may *dynamically leave or join the crowd* without prior notice, and these must be accommodated by *opportunistically* seeking out new resources as they are encountered and having appropriate *fault-tolerance mechanisms* to support mobility.

Honeybee accommodates the above requirements by being proactive and opportunistic, where jobs are 'taken' by nodes rather than 'given to' nodes, as the availability and resourcefulness of each node is unknown a priori, and subject to change any time. For example, if a participating mobile device receives a call, its resourcefulness may decrease, or the user may move away, causing the device to be unavailable. In this work, the device having the job queue representing the task to be completed, is called the *delegator* as it delegates a portion of its task to others. The devices with whom these jobs are shared are referred to as *workers*.

3.1 Target applications

Target applications fall into three categories as given below:

1) Human aided computation is related to enabling collaboration among mobile device users for tasks demanding human specific skills (eg: qualitative classification).
2) Machine computation applications aim to improve the performance and/or conserving resources such as energy, for programs needing extensive computational resources such as memory, battery, and CPU.
3) Applications using Hybrid computations are the ones that are a mix of the two aforementioned categories.

3.2 Job scheduling method

The following characteristics of a mobile edge-cloud need to be considered when scheduling jobs among nodes:

1) *heterogeneity*: since nodes may be of heterogeneous capability and jobs may require varying amounts of resources, job allocation is non-trivial. Optimally stronger nodes should do more work. An expiration mechanism is needed so that stronger nodes can steal expired jobs taken by weaker nodes. Otherwise, if jobs were farmed equally, weak nodes may become bottlenecks.
2) *unknown capability*: since the delegator is unaware of worker capability, it is not possible for the delegator to assign more work to stronger nodes. Exchanging metadata is not effective due to node dynamism, e.g., the node capabilities may change randomly, thereby making the information derived from metadata invalid.
3) *dynamism*: due to mobility and factors such as human intervention and low battery, nodes are prone to failure. Hence the possibility of frequently disconnections and new nodes randomly joining need to be supported, and the overall strategy needs to focus on short term goals and take advantage of opportunities as they arise.

Addressing heterogeneity and unknown device capability: The well-known *work stealing* method [8] can accommodate the first two factors of *heterogeneity* and *unknown capability* given above. This been shown to be an efficient and scalable load balancing method for shared and distributed memory systems [16] in traditional distributed environments [63], and has been used in Cilk ([9], [33]), Parallel XML processing [40], and Energy-efficient Mobile grids [56]. Furthermore, it is able to achieve this without a centralized control, and no prior information about the participating devices. As shown in [33], work stealing is efficient even with different processors with dynamically changing speeds.

Addressing dynamism: To satisfy the third factor of *dynamism*, we have included fault-tolerant mechanisms and also opportunistically attempt to connect to new resources as they are discovered in our model. The dynamic nature of the mobile crowd can cause the following events:

1) a worker's capability changes (e.g., moving away from the delegator while keeping in range, or vice versa).
2) new devices appear within range.
3) a worker device continues to be visible, but becomes non-responsive (e.g., the device stays within range, but the user terminates participation due to low battery).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2560163, IEEE Transactions on Cloud Computing

TABLE 2: Comparing Honeybee with related work

Name	Objective	Underlying framework	Communication protocols	External coordination?	Load-balanced?	Disconnections supported?	Opportunistic?	Job sharing
Honeybee	Performance gain and save energy. Speedups up to 4 when sharing with equally capable devices	None. Implemented from scratch	Wi-FiDirect	No	Yes. via work stealing	Yes	Yes, via periodic resource discovery	Jobs are proactively *taken* by nodes rather than *given* to
Hyrax[42], [3]	Performance gain and save energy. Performance is compared against using servers and not against monolithic execution	Based on Hadoop	Wi-Fi	Yes	Not mentioned	Yes, via Hadoop's node failure handling	No	Name node assigns jobs to slave nodes
MMPI[18]	Performance gain. Maximum speedup with four nodes was 37% for matrix multiplication	Based on MPI	Bluetooth	No	Not supported, but has been suggested	No	No	Uses Master-Slave work farming
Virtual Mobile Cloud[31]	Performance gain and save energy. Performance is less than 1% slower slower than monolithic	Based on Hadoop	Wi-Fi	No	Not mentioned	No. User must be in a stable space	Identify new devices via mobility traces (not implemented)	Workers are given jobs by Master
Serendipity [58]	Performance gain and save energy. Speedups upto 3 when sharing with a stronger device	None	Wi-Fi	No	Not mentioned	Yes. via job expiration	Yes. via exchanging meta-data	Workers are given jobs by Master
Mobile Device Cloud[21]: an emulation testbed	Performance gain and save energy. Gain in both time and energy, up to 50% and 23%	Based on Serendipity [58]	Bluetooth, Wi-Fi, Wi-FiDirect	No	Not mentioned	No	No	Workers are given jobs by Master
MClouds[45] Proposed system, no implementation	Pull data off the cellular data channel & reduce load on backend clouds	None	Wi-Fi	No	Not mentioned	Yes, by monitoring device movement & prior notifications	Yes, via periodic resource discovery	Workers are given jobs by Master
Femto-Cloud [28]	Performance gain	None	Wi-Fi	No	Master schedules jobs based on task complexity and computational capability of the workers	Yes	Yes	Workers are given jobs by Master

4) a worker ceases to be visible (e.g.: moving away, out of the delegator's range).

The first event is addressed by Honeybee's work stealing method as described earlier, since the mechanism automatically adapts to workers' changing circumstances. For example, if the worker's capability increases during execution, it will complete its jobs faster, and steal more jobs from the delegator. Otherwise, if the worker's capability decreased, it will take more time to finish its jobs which may lead to the jobs being stolen by another node. The other three events and Honeybee's methods of addressing them are shown in Figure 2. In the figure, at time t_1 the mobile crowd consists of the delegator n_0 and workers w_1 and w_2, who are already executing their jobs. The event described in item 2 occurs at time $t_1 + \delta$ when a new device w_3 arrives within range of the delegator. Honeybee conducts resource discovery every $t_2 - t_1$ seconds and therefore, the

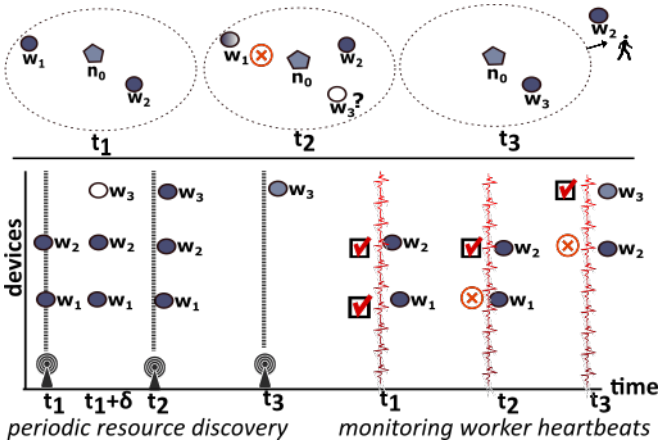

Fig. 2: Handling device mobility and disconnections

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2560163, IEEE Transactions on Cloud Computing

new device w_3 is discovered at time t_2 when the discovery occurs, whereupon Honeybee successfully recruits w_3 as a worker. An event similar to that given in item 3 occurs at time t_2 when w_1 becomes non-responsive. Since w_1 remains within n_0's range, this is not evident from resource discovery. To accurately identify that node w_1 has failed, Honeybee employs *heartbeats* (Section 3.4.4) which alert the delegator whenever a worker becomes non-responsive. As can be seen in Figure 2, Honeybee receives heartbeats from w_1 and w_2 at time t_1, but does not receive w_1's signal at t_2. Hence w_1 is correctly identified as 'dead' and no longer participating in the work sharing, in which case, the work that was stolen by w_1 is added back to the job pool so that someone else may steal them. An event as described in item 4 occurrs at time t_3 when node w_2 moves out of range from the delegator, causing disconnection. This can be identified from resource discovery as well as from the absence of w_2's heartbeat.

3.2.1 Automatic load balancing with work stealing

We employ a modified version of the *work stealing* method [8] for job distribution and load balancing, adapted to better suit an autonomous mobile environment with frequent disconnections. The task to be completed is broken down into a collection of *jobs* stored in the job queue. Jobs do not need to be uniform but must be independent. Upon program initiation, the delegator's local execution thread starts to consume the job queue. The same queue is also consumed by the *steal requests* from worker devices. When a device *steals* jobs from another device, the stealing device is referred to as a *thief* and the other as *victim*. In our previous work[22], [23], all the contents of the job queue were equally distributed among all the workers at the start of the execution, and nodes would begin to *steal* after finishing their own queues. However, this is not ideal for a number of reasons:

- As mobile environments are prone to disconnections, giving a large number of jobs to a worker at once is risky.
- Strong devices steal jobs from weaker devices, and this load balances the work as time progresses. However, stealing incurs transmission costs, especially if the job data is large, so it is important to minimize the number of times a job travels from node to node.
- The number of workers at the start of execution can change throughout execution, and the system must be able to take advantage of new devices encountered.

It is more efficient to let workers steal jobs in small chunks when they are able to, and we have followed this method in our latest model. Overall, the delegator steals from slow workers and fast workers steal from the delegator.

3.3 Honeybee at work: Scenarios

In this section, we provide a walk-through of Honeybee in action, followed by other scenarios where various tasks are parallelized, distributed to and executed on mobile devices.

3.3.1 Transcribing on the train

Language translation and transcription have been successfully crowdsourced via online volunteers recently [65], [49]. YouTube now enables viewers to contribute and review

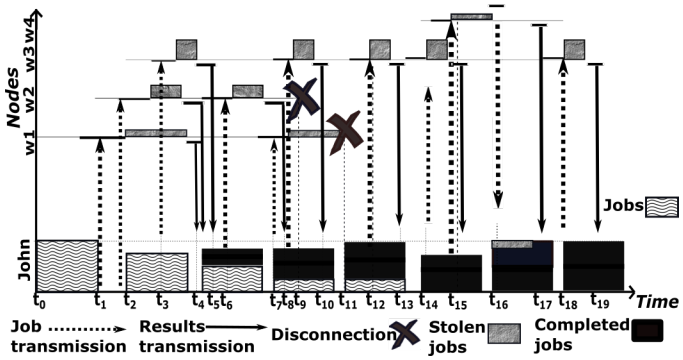

Fig. 3: Five nodes using work stealing

subtitles on some channels[4], and viewers can also request subtitles for videos. The Google Translate Community[5] also enables volunteers to translate and review phrases. This concept of crowd-powered translation and transcription via the Internet can be extended to allow edge-cloud enabled crowdsourcing to utilise the linguistic knowledge of local users. For example, consider a tourist John Doe travelling on a train in a foreign country. He has several local films in his tablet, but they are in the country's native language, which he does not understand. Unfortunately, these films do not have subtitles. Looking around his train, John realizes that the local passengers could help with his dilemma. But how can he enlist them to solve his problem? Honeybee can provide a solution as follows;

a) First, John uses his mobile device to split the video into chunks of equal duration, thereby initiating his job queue consisting of n jobs. This is illustrated in Figure 3, where the total job queue is initialized at time $t0$.
b) Next, he scans the vicinity via the tablet's Wi-Fi Direct, looking for available worker devices.
c) Once a worker is connected, it steals k jobs from the queue. In Figure 3, the delegator transmits stolen jobs to workers w_1 and w_2 at times t_1 and t_2 respectively.
d) Using a mixture of human intelligence and machine resources, each worker device processes its video chunks by adding English subtitles to their jobs. For example, w_1 executes its first batch of jobs starting from time t_2.
e) As soon as a worker device completes m jobs, it sends the results to John. To minimize communication costs, workers only send back the subtitles, with markers indicating the time intervals they correspond to. In Figure 3, John (delegator) receives results from workers w_1, and w_2, at time t_4 and from worker w_3 at time t_5.
f) When a worker finishes its jobs, it tries to steal more jobs from John's main job queue as can be seen at time t_{16}. Here, the delegator (since John does not have the knowledge to add subtitles himself) does not work, and has given all the jobs to workers. When w_3, having finished all of its jobs, tries to steal more from the delegator, the delegator must then try to steal from some other slow worker on behalf of w_3. For example, the delegator steals jobs from w_4 at time t_{16}, and sends them to w_3 at time

4. https://www.youtube.com/watch?v=b9cKgwnFIAw
5. https://translate.google.com/community

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2560163, IEEE Transactions on Cloud Computing

t_{18}. When John receives all completed jobs by time t_{19}, he sends a termination signal to all the workers.

g) As this occurs on board a train, passengers are frequently getting on and off, including the ones owning the workers. Whenever a worker disconnects, John's tablet gets to know several seconds later from the missing heartbeat. In such a case, jobs stolen by the missing worker will be added back to the queue, making them available for another to steal. Such a scenario occurs at times t_9 and t_{11} when workers w_2 and w_1 disconnect. In the case of w_2, the disconnection happens after it has sent the completed jobs to the delegator, and before any new jobs were stolen. Therefore, no further action is needed from the delegator. However, when w_1's disconnection is sensed, jobs had already been stolen by w_1. When the delegator decides that w_1 has died, it looks up the jobs that were given to w_1, and adds them back to the queue. Similarly, new workers may also join as passengers get on the train as John is continuously scanning for new resources every r seconds. Table 3 summarises the job characteristics.

TABLE 3: Job characteristics: Video transcribing on the train

Objective	Execute program: delegator is incapable of doing the task by itself and ∴ delegator does not work.
Worker encounters	High
Disconnections	High
Job format	Video files
Result	In string format

3.3.2 Facial recognition at the workplace

Even a resource rich environment can sometimes be unusable depending on the type of work and the location of data. For example, take the case of Jane at her office equipped with many PCs. She has hundreds of photographs on her camera phone, taken at a recent office party. Her friend Mary asks to send her all the photographs where Mary appears. As Jane is unable to connect her phone to a computer, she considers using her the device to run a facial recognition app that compares each of the images with a photograph of Mary and filters the ones containing her face. However facial recognition algorithms are costly, and processing a large number of photographs could take a substantial amount of time, freeze the device and drain the battery. These challenges can be addressed by offloading/crowdsourcing the image processing task to external computing resources, as explored in [59], [38], [44] and implemented in projects such as GeoTag-X[6] and Galaxy Zoo[7]. However, the aforementioned approaches need remote clouds and/or cloudlets, neither of which are available to Jane. Therefore Jane employs Honeybee to share the task with mobile devices belonging to her colleagues as follows:

a) The job queue has all photos taken on the specific day.
b) Each job has an image file (to be compared with Mary's image). The job characteristics are given in Table 4.
c) 'Workers' would be Jane's colleagues' mobile devices.
d) Jane's device begins to run the facial recognition program on the job queue, and connects to the workers and transmits jobs from the same queue in parallel.
e) Workers get image files from the job queue, and Mary's photo to compare against. As they finish their jobs, they send the results to the delgator as Strings, indicating the file name and whether it was positive or negative.
f) This will continue as described in detail in Section 3.3.1, until all the jobs are completed.
g) Jane's device would then copy all the images with a positive result to a separate folder, ready for Jane to transfer to Mary's device via Bluetooth or Wi-Fi.

We experimentally evaluate a similar task in Section 5.

TABLE 4: Job Characteristics: Facial recognition in the office

Objective	Obtain speedup: Delegator can do job by itself, but requires too much time, battery and probability of crashing is high. ∴ delegator does part of the job.
Worker encounters	High: assuming a typical office environment
Disconnections	Medium: assuming some office workers move in and out of the space frequently, while some work steadily at their desks
Job format	Image files
Result	In string format

3.3.3 Mandelbrot set generation in an underpriviledged classroom

Cloud computing is already being applied in a number of developing countries [43], but there are some practical issues such as frequent power interruptions, and the need for robust broadband infrastructure [60]. However, the usage of mobile phones and mobile data have increased over the years, especially in the Middle East, Asia Pacific and Africa [1]. Hence, resource sharing via mobile devices can be greatly beneficial to such regions by empowering them while spending little cost on upfront investments and infrastructure. For example, let us take the case of an underprivileged classroom, where the lesson is on the Mandelbrot set. The teacher John wants to demonstrate how different variables affect the set with aid of computers, but does not have access to a laptop or PC. The school has some tablets and most of the teachers and some students have smartphones, but running Mandelbrot set generation on a smartphone is not practical due to the time constraints of the lesson and issues with battery drain and system failure. Therefore, John decides to employ Honeybee as follows:

a) The Mandelbrot set is given as x rows and y columns, and each job represents one row, giving a total of x jobs.
b) The potential worker devices would be the mobile devices of the teachers and students in the vicinity.
c) The job descriptions are specified as strings and the results are transmitted from the workers as integer arrays. The job characteristics are illustrated in Table 5.

This task has been implemented using Honeybee and experimental results will be discussed in Section 5.

3.3.4 Other applications

Similar applications can be implemented using the same principles in other areas such as disaster management,

6. http://geotagx.org/
7. http://www.galaxyzoo.org/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2560163, IEEE Transactions on Cloud Computing

TABLE 5: Job Characteristics: Mandelbrot set in the classroom

Objective	Obtain speedup: Delegator can do job by itself, but will require too much time, battery and probability of crashing is high. ∴ delegator does part of the job.
Worker encounters	Medium
Disconnections	Low: since students and teachers tend to stay within a classroom, school boundaries within a given time
Job format	String
Result	As integer arrays

sensor data processing programs, and medical data analysis. For example, use of external devices for processing bio-signals has already been discussed in systems such as MobiHealth [32], and remote ECG data analysis [50] using remote servers. However, instead of transmitting sensor data from medical equipment over the Internet, Honeybee would use the patients' surrounding devices to analyse the data.

3.4 Strategies for efficiency optimisation

Several optimisations have been performed for efficiency, including using Wi-Fi Direct as the mode of communication, setting the *steal limit* and the mechanisms of *job expiry*, *heartbeats* and *periodic resource discovery*.

3.4.1 P2P communication using Wi-Fi Direct

Wi-Fi Direct is used as the mode of communication to achieve our objective of minimising transmission delays (Table 1). Wi-Fi Direct allows P2P Wi-Fi connections between 'Wi-Fi CERTIFIED®' devices without the need for Wi-Fi APs [61], at Wi-Fi speeds. Its connection process has three main stages; the search and discovery stage is the first, and is followed by the Group Ownership negotiation stage. Here, the P2P group is formed consisting of one P2P group owner (GO) which implements AP-like functionalities, and P2P clients. Since these roles are dynamic, devices need to negotiate their roles prior to establishing the group [10]. The device with the highest *Intent* value (a number from 0 to 15) is designated as the GO. The allocation of IP addresses is the final stage, where the GO provides the clients with IP addresses. As long as the hardware of GO supports Wi-Fi Direct, legacy devices with upgraded software can function as P2P clients. As shown in Table 6, Wi-Fi Direct outperforms Bluetooth in speed, range and security.

TABLE 6: Comparing Wi-Fi Direct with Bluetooth

	Bluetooth 4.0	Wi-Fi Direct
Speed	721.2 kbps (basic), 2.1 Mbps (enhanced) & high speed operation up to 54 Mbps [7]	up to 250Mbps
Range	up to 100 m	up to 200 m
Security	AES 128-bit encryption	WPA2 security
Power	low-energy technology	two power saving modes
Availability	widely available	not as widely available

3.4.2 Job expiry

The *job expiry time* is the minimum amount of time a worker would be allowed to complete a given job. This mechanism is needed to prevent stronger devices waiting indefinitely for a weaker device attempting a very intensive task. After a node starts running a job, it cannot be stolen. Hence the delegator needs to decide whether or not to term the job/s as expired and add them back to the job queue, which would give a chance for any other node to complete it. Based on the time the job/s were stolen, the *oldest* jobs would be termed as expired. The only time a delegator expires jobs is after an unsuccessful steal attempt and it's job queue is exhausted. For example, the scenario in Figure 4, shows the number of jobs left in the delegator's and a worker's job queues over time. For ease of illustration only one worker is shown in the figure, although other workers exist. At T_0, the delegator has m number of jobs in its queue. At time T_e, $worker_e$ successfully steals j jobs from the delegator. These are received by $worker_e$ at time $T_e + \delta$ and $worker_e$ starts executing the stolen jobs immediately. However, from the delegator's point of view, the jobs were stolen at time T_e, and therefore, logs the jobs' stolen time as T_e. Time progresses, and the delegator finishes its own job queue at time T_d. At this point, the delegator attempts to steal some jobs from another worker (not $worker_e$) and is able to add k stolen jobs to the queue at time $T_d + \theta$. By time T_ω, the delegator completes the aforementioned k number of jobs as well. Once more, the delegator attempts to steal, but receives a negative answer at time T_s. The delegator examines the stolen job list after each unsuccessful steal and at time T_s, the delegator consults the list of jobs that have been stolen, but whose results have not been returned. The oldest jobs left are then identified to be the j number of jobs that were stolen by $worker_e$, are added back to delegator's queue, and are completed by time T_f. There may be cases when the node $worker_e$ is not actually a weak node, but the j jobs are extremely intensive such that it is more time consuming than all of the other jobs accumulated. However, even in that scenario *expiring* the jobs would not harm the overall performance as the task would be finished as soon as either node finishes the job, and the nodes in the system have no other jobs to work on.

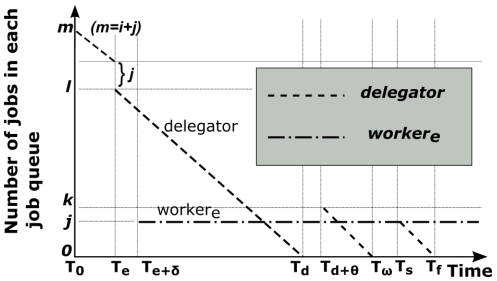

Fig. 4: Expiring oldest jobs

3.4.3 Steal limit

Each device has a preset *steal limit* s, and can be described as the number of jobs a nodes keeps in reserve when it receives a steal request from another node. The steal limit is job specific and the default value can be overridden to suit the needs of the application. As opposed to jobs simply being transferred among devices in an unending manner, this ensures a device will not starve and computations will eventually terminate.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2560163, IEEE Transactions on Cloud Computing

3.4.4 Worker heartbeat

The dynamic nature of mobile edge-clouds will incur frequent and unpredictable disconnections. If the disconnections are unidentified, the delegator may wait unnecessarily for the return of stolen jobs. To address this, each worker sends a periodic signal to indicate that it is *alive*. If the worker had sent results to the delegator, or acknowledged a job transmission, within that time period they are also counted as *heartbeats*. If the delegator cannot hear a worker heartbeat for m consecutive checks, the delegator deems that worker is *dead*, i.e, either moved away or lost connectivity, and adds the respective stolen jobs back to the queue.

3.4.5 Periodic resource discovery

As much as random disconnections are an inherent attribute of a mobile resource cloud, so too are random device encounters leading to connections. To support opportunistic resource connections as and when they become available, a periodic resource discovery is done by the delegator every r seconds, and carried out till the task completes.

3.5 Conditions for speedup

The probability of speedup depends on a number of factors;

1) Parallelization overhead: the additional time spent on co-ordination (initiating the job queue by breaking the total task down to jobs, maintaining a thread pool, synchronization, handling incoming messages from workers, monitoring worker health) adds extra costs. Communication costs (transmitting jobs to workers and receiving results) are not included as they are included in the workers' job completion time (see factor 5 below).
2) The serial task running time should not be too short: for tasks with very short running times, parallelising and job distribution only add extra costs. The time to complete a parallelised task on Honeybee depends on the parallelisation overheads, running time of the delegator thread doing useful work, and the running time of delegator's communication thread handling worker transmissions. Depending on node capability, and communication constraints, either thread may finish first. To match serial performance, the workers must make up for at least the parallelisation overheads. To gain speedups, the monolithic task time must at least be greater than $t_{com} + c$ to amortize the delegator's communication and parallelisation costs (where t_{com} is the time to establish connections with the workers plus the time to transfer jobs and results to and from workers, and c is the delegator's parallelisation cost).
3) Individual job size: once the total task is broken down to individual jobs, each job size should not be too large so that transmitting them to workers will incur substantial communication costs. For example, if the time to execute a given job j_x on the delegator is time t_x and the communication time to transfer the said job to a given worker, and its results back to the delegator is time t_y, then it must be that $t_x > t_y$.
4) Jobs must be independent: the current implementation of Honeybee can only handle independent jobs.
5) The *capability* of worker devices: by this we refer not to the CPU speeds alone, but overall how much work a worker can do in a given time. Therefore, a worker that has a powerful CPU, but has low availability would not be considered as possessing high *capability*.

From the above list, items 1 to 4 can be determined prior to job execution, and can be regarded as 'known'. Items 1 to 3 are job dependent and item 4 depends on the implementation of the Honeybee framework. But the last item regarding worker capability is impossible to know a priori. Worker capability can further be expressed in terms of the amount of work a worker completes compared to the delegator.

3.6 Upper and lower bounds for speedup

Let us denote each device as n_i, where the delegator would be denoted as n_1, and the time taken to complete m jobs on n_1 as t_1. The time taken to receive, complete and send the results of m jobs on a worker device n_i can be given as t_i, where $i > 1$. To express the 'capability' of worker n_i in terms of n_1, the relationship between t_1 and t_i where $i > 1$ needs to be examined. Let us say there exists a non negative constant k_i for each n_i device such that k_i is the relative power of n_i compared to n_1, and given as follows:

$$\frac{t_i}{t_1} = k_i \tag{1}$$

Let us say that a task consisting of l jobs were completed on this system containing nodes from 1 to f. If the number of jobs completed by each node n_i can be given by h_i, then the total number of jobs completed by the delegator node n_1 is h_1, where $l > h_1 > 0$. Then, the total number of jobs completed by all the worker nodes can be given by $\sum_{i=2}^{f} h_i = l - h_1$. Depending on the values of h_1 and $l - h_1$, and assuming that all the jobs were equal, the capability of the worker devices can be compared to the delegator. Therefore, the worst case scenario for Speedup is when the collective capability of workers is weaker than the delegator; i.e. $h_1 > l - h_1$. Here, Speedup S is defined as the comparison between the time taken to complete a task using Honeybee versus the time taken to execute the task monolithically (the 'monolithic version' refers to the task without any of the parallelizing components). Therefore, when t_M is the time to complete the 'monolithic version' of the task on delegator node n_1, and t_p is the total time to complete the parallelized version using Honeybee, S can be given as,

$$S = \frac{t_M}{t_p} \tag{2}$$

We gave an upper bound for Speedups in our previous work [23] as $S = 1 + \frac{1}{k_2} + \frac{1}{k_3} + ... + \frac{1}{k_f}$, assuming equal jobs and ignoring overheads. Since the overheads are non-negligible, and jobs are not always guaranteed to be equal, the actual speedups would be less than this value. Here, we derive a lower bound for speedup, considering the worst case scenario discussed above. We assume that the delegator will be doing part of the job, although it is not always the case (e.g., the scenario in Section 3.3.1). If the delegator is unable to contribute to the work, there could not be a comparison for Speedup anyway. In the worst case scenario, the collective capability of worker devices is infinitely less than that of the delegator. Let us assume that in the extreme case, the collective capability of the workers is so small compared to the delegator, that their contribution is non-existent. This is similar to the case when

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2560163, IEEE Transactions on Cloud Computing

the delegator executes the parallelised version, but fails to find any worker nodes during the entire course of the execution. There are still overheads with no workers, such as parallelising costs and searching for workers periodically. In this case, if the time to complete the parallelised version only using delegator n_1'resources is given by t_o, then t_o is greater than t_M. Since the monolithic version is devoid of the parallelisation overhead, t_o is given by,

$$t_o = t_M + c, c \text{ is the parallelizing overhead for } n_1 \quad (3)$$

Figures 5a and 5b show the scenarios for task times t_M and t_o. As explained in Section 3.4.2, the effect of extremely weak nodes is dealt with by expiring the oldest jobs after the delegator exhausts its own queue. Let us say the time for the delegator to complete the jobs, with extremely weak nodes that do not contribute to the work in any way, is time $t_{\bar{o}}$ (Figure 5c), i.e.,

$$t_{\bar{o}} = t_o + e, \text{where } e \text{ is the job expiration cost for } n_1 \quad (4)$$

Hence the worst case job completion time t_{worst} when all workers are infinitely weaker than the delegator is,

$$t_{worst} = t_{\bar{o}} = t_M + c + e \quad (5)$$

$\therefore$ the lower bound for speedup can be derived as,

$$S \geq \frac{t_M}{t_M + c + e} \quad (6)$$

In summary, the collective capability must amortise the parellelisation cost, as experimentally illustrated in Section 5.

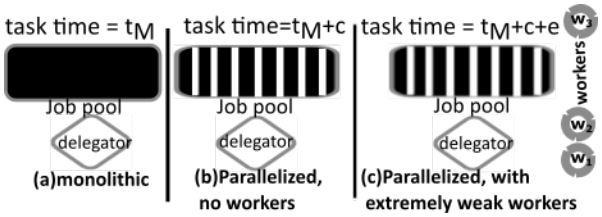

Fig. 5: Comparing the task time for three scenarios

4 Implementation

Honeybee is implemented on Android, using Wi-Fi Direct as the communication protocol. Application developers can use the methods and interfaces provided by the framework for writing work sharing mobile apps. As shown in Figure 6, the framework contains three main components responsible for the main areas of Application interfacing, Job Handling, and Communication.

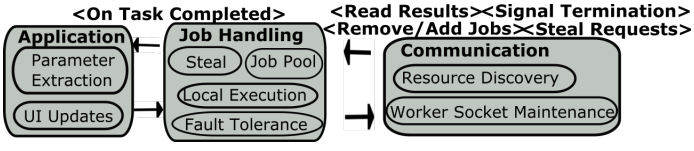

Fig. 6: Main Components in the Delegator

4.1 The application component

The Application layer methods interfaces between application specific code and the core structure. At the starting point of execution, the framework extracts the application specific parameters via the *AppRequest* interface that provides abstractions to represent the task as a list of jobs.

4.2 The job handling component

The initialization of the job pool begins with processing the AppRequest object. Once the AppRequest task is broken into sub tasks, or jobs, they are stored in two data structures: in an array `allJobs` that is not modified throughout execution and in a ConcurrentLinkedQueue `jobList`, which is subject to change dynamically, as jobs are removed and added from accessing threads. The array is maintained for validation purposes. During the lifetime of the program, `jobList` is accessed by several processes as given below:

1) Delegator's local job execution thread: Jobs are consumed by polling the `jobList` from the head.
2) Steal request threads from workers: as steal requests arrive from the Communication component, they are processed by the Job Handling component which creates `Callable` instances to consume jobs in k chunks from `jobList`. More than one steal requests may arrive and be processed at the same time.
3) Threads carrying stolen jobs from workers: When the delegator steals jobs from workers, they are passed from the Communication component to the Job handling component, where the job parameters are decoded and assembled into jobs, and added to the `jobList`.
4) Fault tolerance threads: as fault tolerance mechanisms, jobs that were already assigned to workers may presumed 'lost' and be added back to `jobList`. This takes place when jobs expire (Section 3.4.2), or worker heartbeats are missed (Section 3.4.4).

In cases (3) and (4) above, when jobs are added, new local execution threads are spawned as *Runnable* tasks and added to a *single thread pool*, thereby ensuring that only one local execution thread is running at a given time. These are then executed locally as described in case (1).

4.3 The communication component

Potential workers are identified by running resource discovery every t seconds. Whenever a new resource is detected, the user has the choice to initiate a connection. For each successful worker connection, a reading thread is kept alive throughout the lifetime of the connection as the delegator needs to receive various messages from the workers at intermittent intervals. The messages expected to be received and written by the delegator are summarized in Table 7.

TABLE 7: Types of I/O messages handled by the Delegator

Read	Write
1. Steal requests by workers	1. Jobs stolen successfully by workers
2. Workers' acknowledgement of receiving job data	2. Reply to unsuccessful steal attempts by workers (when the delegator does not have any jobs for workers to steal)
3. Negative replies to steal attempts by the delegator (when a worker does not have any jobs)	3. Steal requests from delegator (when the delegator attempts to steal from others)
4. Stolen jobs in cases of successful steals	4. Termination signal sent to workers once delegator verifies all jobs have been completed.
5. Results sent by workers	
6. Worker heartbeats	

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2560163, IEEE Transactions on Cloud Computing

5 Experimental Evaluation

This section evaluates the Honeybee algorithm, focusing on speedup, and the best and worst case scenarios of machine-centric computation. Human-centric computation with an app for collaborative photography is discussed in our previous work [23], [22].

Testbed: A heterogeneous testbed of smartphones were used in the evaluations of the previous phases of our model in [25], [22], [23], representing low to mid-end mobile devices. For this phase, we chose a testbed of 9 Nexus 7 (2012) tablets, representing high-end mobile devices, and each worker device was placed within 1m of the delegator. As work-stealing was shown to be effective on a heterogeneous set of devices in [23], in this phase, we use a homogeneous testbed to further extend the range of testing, and to simplify the analysis of results. To ensure objective comparison, the same device was used as the delegator (Android 4.2.2) for all of the tests. Performance using Honeybee was evaluated against performance of the monolithic versions. In particular, the evaluation objectives are: 1) examine the speedups for a fixed task size for varying number of devices; 2) examine the speedups for fixed numbers of devices for varying task sizes; 3) examine energy consumption for a fixed task size for varying number of devices; 4) experimentally demonstrate the mathematical lower bound on speedup; and 5) examine program behaviour with random disconnections.The results were obtained from two applications implemented using the Honeybee API, as given below:

Distributed face detection: Face detection requires a large amount of CPU and memory. Running face detection on a considerable number of images is usually very slow, and can cause the application to be non-responsive, or even cause OutOfMemoryExceptions and incur high energy costs. Using Honeybee, we aim to address these issues by sharing the resource intensive computations with other devices. In this application, Android's native face detection algorithms are executed on a collection of photographs. This collection contains 30 unique image files with a total size of 8.4 MB. In order to achieve uniform comparisons for different job pool sizes, we duplicate the same files for job pools of 120, 240, 480, 960, 1920, 3840, and 4800. These images are stored in the delegator (a Nexus 7) at the start of execution.

Distributed Mandelbrot set generation: In the context of the Mandelbrot set, jobs represent rows of a 300 x 300 Mandelbrot image. These applications were chosen for their different characteristics, as listed in Table 8.

TABLE 8: Applications' characteristics

Name	Type	Inputs	Outputs
Face detection	Machine centric: CPU and memory intensive	Images-large data size	String with image name and # faces detected, small data size.
Mandelbrot	Machine centric: CPU intensive	Strings-Small data size	Integer arrays, large data size.

5.1 Results & Discussion

Figures 7, 8 illustrate the performance results of experiments for the Face detection app and the Mandelbrot app using Honeybee. All tests were repeated at least three times. These are summarised in Table 9 with the average speedup, standard deviation of speedups and the confidence (Cd) for significance value of 0.05.

TABLE 9: Results with standard deviation and confidence

#Devices	FA	Avg. S	Std	Cd	MA	Avg S	Std	Cd
2	C	1.680	0.082	0.093	N	1.621	0.096	0.094
3	E	2.247	0.074	0.084	D	2.145	0.114	0.129
4	M	2.819	0.058	0.066	E	2.683	0.042	0.042
5	A	3.213	0.109	0.123	L	3.083	0.117	0.115
6	T	3.879	0.164	0.186	B	3.374	0.056	0.049
7	C	4.012	0.059	0.066	R	3.471	0.084	0.074
8	H	-	-	-	O	3.655	0.095	0.094
9		-	-	-	T	3.724	0.057	0.064

5.1.1 Performance gain

Both applications were tested for speedups for a fixed task size while varying the number of devices. Figure 7a gives the face detection performance results for 960 images. As can be seen, the speedup is proportional to the number of devices and the maximum average speedup observed was 4.012 for 7 devices. Results from Mandelbrot set generation show a similar trend in Figure 7b where the maximum average speedup was 3.724. Figure 7c shows the speedups for Face detection using varying numbers of jobs versus a fixed number of devices. Comparing the results for both 2 and 3 devices, in both cases the speedup increases proportionally to the task size. Figures 7dand 7e show the percentage and amount of time saved for Face detection, and Figure 7f shows the percentage of time saved for Mandelbrot set generation. As can be seen from all three graphs, the time saved is proportional to task size (total number of jobs) and amount of resources. From Figures 7a, 7b, 7d and 7f, it is clear that the speedup plateaus after reaching the maximum speedup value. Overheads due to maintaining connections and parallelisation could be the reason for this. From a communication perspective, as the number of concurrent connections increase, the workers must compete for the same channel, thereby reducing the data rate for each device. Furthermore, the delegator must manage more concurrent threads as more and more workers connect, which can slow the delegator. This is evident from Figures 7a, 7b, and 7f, where the rate of performance increase gradually slows down as number of devices increase (discussed further in section 5.1.6 according to data in Table 12). Implementing a hierarchical structure may help to overcome this barrier. In Figure 8a speedups are mapped against the percentage of jobs done by the delegator for Face detection. Here it is clear that maximum speedups are obtained when the delegator does the least amount of work.

5.1.2 Effect of Wi-Fi Direct

When comparing these results with the results for Face detection in our previous work in Phase II [23] using Bluetooth, there is a marked improvement in communication costs. In this phase, using Wi-Fi Direct, the average data rates of the workers and the delagator are 10.444 Mbits/s, and 14.262 Mbits/s respectively. In contrast, using Bluetooth 3.0, the average job transfer rate of the delegator for the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2560163, IEEE Transactions on Cloud Computing

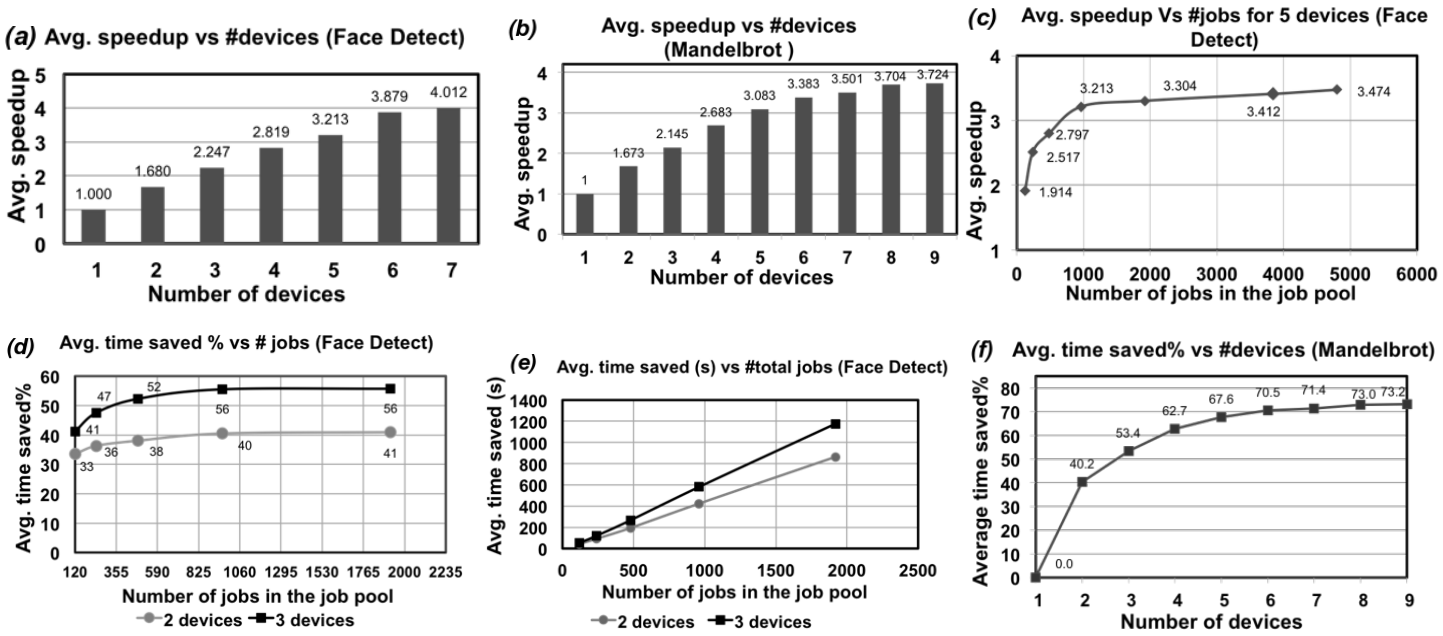

Fig. 7: Experimental results on Face Detection and Mandelbrot set

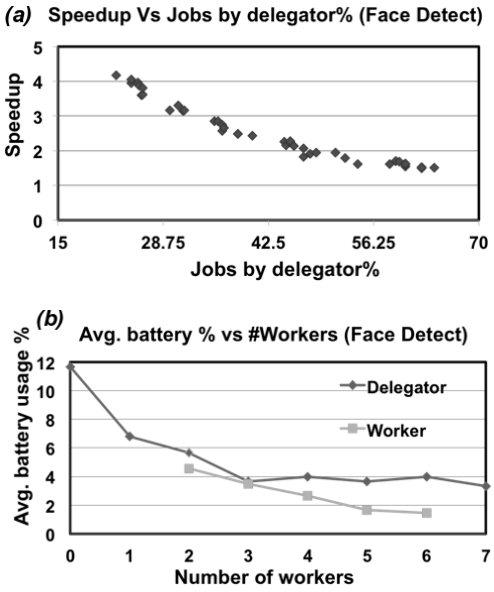

Fig. 8: Experimental results continued

same app was 2.124 Mbits/s. The Wi-Fi Direct speeds observed appear to be much less than its maximum speed (Table 6). This may be caused by maintaining multiple connections. A main drawback of Wi-Fi Direct is its long group formation time compared to Bluetooth. To amortise this, the groups need to have a significant life time, and/or have heavy data communication. It was also found that only a maximum of 8 Wi-Fi Direct connections were supported per each Nexus 7 device, limiting tests to a maximum of 9 devices. New D2D technologies like LTE-Direct[8] may be able to solve these problems. Compared to Wi-Fi Direct, LTE-Direct has a greater range (up to 500m), has a faster one-step connection process, and its 'always-on' discovery method enables it to discover more peers continuously. Hence using LTE-Direct for Honeybee can give support for more workers over a greater range, thus giving more mobility and better performance.

8. www.qualcomm.com/invention/research/projects/lte-direct

5.1.3 Energy consumption

The energy consumption was measured via the Android battery API. Battery levels of the delegator were taken just before the program start and just after program end. Our experiments with the Face detection app for 1920 images are summarized in Figure 8b showing the battery drain of the delegator and the workers. As can be seen, the energy usage of the delegator is almost halved from 11.67 % to 6.80% with just one worker. The average battery usage per each worker is also reduced as the number of workers increase, and the energy usage per worker is less than the delegator's use. However, as the number of workers increase, the energy saving does not increase drastically, possibly owing to costs related to parallelisation and maintaining connections.

5.1.4 Lower bound

We tested the worst case scenario using Honyebee as discussed in Section 3.6 and measured the cost of job expiry. We emulated 'weak workers' by running an infinite loop inside the worker devices, thereby making them infinitely slower than the delegator. In this case, although the delegator is connected to workers and the workers have stolen some work, the workers are so slow that the delegator expires the stolen jobs and completes all of the jobs by itself. As can be seen from Table 10, the experiments gave an average performance loss of 9.9% for $t_{\bar{o}}$. In contrast, the average performance loss for t_o (as discussed in Section 3.6, Equation 3) was 7.8%, which gives a 2.1% difference in cost between

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2560163, IEEE Transactions on Cloud Computing

TABLE 10: Speedups for worst case senario in several setups

Setup	Speedup	Performance loss
7 workers	0.9012	(1-0.9012) * 100 = 9.8741%
4 workers	0.8906	(1-0.8906) * 100 = 10.9439%
3 workers	0.8956	(1-0.8956) * 100 = 10.4382%
2 workers	0.9096	(1-0.9096) * 100 = 9.0342%
1 worker	0.9070	(1-0.9070) * 100 = 9.2978%

the scenarios discussed in Figures 5b and 5c, possibly due to the cost of establishing connections and job expiry.

5.1.5 Random disconnections

To test the effectiveness of fault tolerance mechanisms handling random disconnections, the delegator was programmatically forced to disconnect its workers at a 10 second interval until no workers remained. This was tested with a setup of 4 workers and the disconnection process was commenced after all 4 workers had started working. The lost jobs were re-assigned to the delegator and the program finished with a speedup despite the disconnections. The results of this scenario with 4 workers is similar to having 1 consistent worker as given below in Table 11.

TABLE 11: Speedup with random disconnections

Setup	Speedup	Jobs by Delegator
4 workers randomly disconnected	1.696	63.333%
1 consistent worker	1.673	63.889%

5.1.6 Device busyness

The efficiency of the system depends on minimising the idle time of the participating nodes by keeping them busy doing useful work. However, bottlenecks in transmission and multi-threading can cause idling. To investigate this, data gathered from the test runs of the Face detection app for 1920 jobs were analysed. Table 12 compares data from three configurations: 1, 3, and 7 workers. Each device's computation time and reading time were measured and given as percentages of its total program time. For example, for 1 worker, the worker's average computation time was 73.53% of its total time. As can be seen from Table 12, the average computation time of a worker decreases significantly as the number of workers increase. Although this trend is also evident in the delegator's computation time, the decrease is very slight. However, the average reading time increases for more workers, despite the decrease in the data being read by each one. As the delegator needs to communicate with and transfer jobs to more and more workers, the time available to each worker can be less. In the case of Face detection, the majority of transmission time is spent on transmitting the jobs (images) from the delegator to workers. The data observed in Table 12 suggests that as the number of workers increase, each worker needs to wait a greater length of time to receive its jobs, thereby decreasing time spent on calculation (useful work). This behaviour is also evident in the speedups as discussed in Section 5.1.1.

5.1.7 Data movement

The same test results discussed in previous section 5.1.6 were examined to check the movement of data within the participating nodes. The experiments were run on 1920 jobs, which translates to a set of 1920 image files, with a total job data size of 538.4 MB. The jobs were originally on the delegator, but moved to workers during the course of execution, as stealing occurred. Table 13 illustrates the amount of data that were transmitted from delegator and the sum of data read by all the workers. The amount of transmitted data does not exceed the actual job data size of 538.4 MB in any of the 3 configurations. The percentage of data transmitted increases from 35.34% to 66.02% as workers are increased from 1 to 7, showing that a higher number of offloading occurred with the addition of workers. Also data was not moved unnecessarily among devices.

TABLE 12: Breakdown of test results for Face detection

# workers	Avg. delegator computation time	Avg. worker computation time	Avg. worker reading time	Avg. data read by each worker
1	99.96%	73.53%	8.53%	190.24MB
3	99.95%	67.78%	11.19%	106.13MB
7	99.93%	57.31%	12.92%	56.43MB

TABLE 13: The movement of data in Face detection app

# workers	Avg. data written by delegator	Avg. total data read by workers	Avg. offloaded data
1	190.24 MB	190.24 MB	35.34%
3	291.26 MB	291.26 MB	54.10%
7	355.44 MB	355.44 MB	66.02%

6 Conclusions & Future directions

We present the following conclusions. Firstly, work sharing among an autonomous local mobile device crowd is a viable method to achieve speedups and save energy. The addition of new resources up to an optimal amount, can yield increased speedups and power savings. Secondly, a generalized framework can be used for abstracting methods and enabling parameterisation for different types of tasks made of independent jobs. Thirdly, inherent challenges of mobile computing such as random disconnections, having no prior information on participating nodes, and frequent fluctuations in resource availability can be successfully accommodated via fault tolerance methods and work stealing mechanisms.

The Honeybee model caters to tasks that can be decomposed into independent jobs. Many crowd computing tasks for mobile devices are suited to this model, for e.g., video transcribing (Section 3.3.1), language translation, medical data analysis (Section 3.3.4), face detection (Section 3.3.2) and mathematical demonstrations (Section 3.3.3). However, there are other tasks that cannot be easily decomposed into independent jobs. Work done by Agrawal et al. shows that work stealing can be further enhanced for dependent jobs[4] and we aim to work in this area in the future. Incentive management and security are important for the deployment of successful mobile crowd applications. However, designing a comprehensive and realistic incentive scheme for mobile crowd computing applications requires further research in collaboration with policy, legal and economics scholars [5], as does providing security and trust mechanisms. As the main focus of this paper was performance gain and energy conservation, these two areas were out of scope. For this

work, we have built Honeybee with the assumption that an incentive system and a secure environment are already in place. Future work could be possible in designing a secure platform for mobile crowd computing applications, supporting incentive management. Moreover, this work focused on the evaluation of machine-centric computation. However, as discussed in Section 3.3, applications that employ human intelligence are also feasible using the Honeybee model. For example, the face detection app in Section 5 can be modified so that human intelligence is used to identify the faces detected by the machine. We aim to extend our evaluations to focus on this aspect, using additional criteria such as accuracy and usability in our future work. Furthermore, as observed in our experiments, the performance gain plateaus as the number of worker nodes increase due to the additional costs that occur when a single device (delegator as P2P group owner) maintains multiple connections. To overcome this and scale up, we plan to extend Honeybee to support other topologies and initial experiments in [39], where an early version of the Honeybee model was extended to support hierarchical Bluetooth connections, show consistent speedups using a linear topology, with an intermediate node functioning both as a worker and a delegator. For this approach, a combination of Bluetooth and Wi-Fi Direct in alternate hierarchical layers can be explored as Wi-Fi direct does not support multiple Wi-Fi direct groups. We also plan to experiment with latest D2D technologies such as LTE-Direct to improve performance. In addition, the experiments in this paper were performed in a controlled setting. We plan to extend these tests to more realistic scenarios by using mobility patterns to simulate churn.

REFERENCES

[1] Cisco visual networking index: Global mobile data traffic forecast update. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-520862.html.

[2] DARPA Creates Cloud Using Smartphones. http://www.informationweek.com/mobile/darpa-creates-cloud-using-smartphones/d/d-id/1111323.

[3] The hyrax project. http://hyrax.dcc.fc.up.pt/.

[4] K. Agrawal, C.E. Leiserson, and J. Sukha. Executing task graphs using work-stealing. In *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1–12, April 2010.

[5] M. S. Bernstein. Crowd-powered systems. *KI - Künstliche Intelligenz*, 27(1):69–73, 2013.

[6] K. Bhardwaj, S. Sreepathy, A. Gavrilovska, and K. Schwan. Ecc: Edge cloud composites. In *Proceedings of 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, pages 38–47, 2014.

[7] Bluetooth. Specification of the bluetooth system version 4.1. https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=282159, December 2013. Accessed: 25/06/2014.

[8] R. Blumofe and C. Leiserson. Scheduling multithreaded computations by work stealing. *J. ACM*, 46(5):720–748, 1999.

[9] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou. Cilk: an efficient multithreaded runtime system. *SIGPLAN Not.*, 30:207–216, August 1995.

[10] D. Camps-Mur, A. Garcia-Saavedra, and P. Serrano. Device-to-device communications with wi-fi direct: overview and experimentation. *Wireless Communications, IEEE*, 20(3):96–104, June 2013.

[11] Z. Chen, R. Fu, Z. Zhao, Z. Liu, L. Xia, L. Chen, P. Cheng, C. C. Cao, Y. Tong, and C. J. Zhang. gMission: a general spatial crowdsourcing platform. *Proceedings of the VLDB Endowment*, 7(13):1629–1632, 2014.

[12] B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proc. of the 6th conference on Computer systems*, EuroSys, pages 301–314, 2011.

[13] B. Chun and P. Maniatis. Dynamically partitioning applications between weak devices and clouds. In *Proc. of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, MCS, pages 71–75, New York, USA, 2010. ACM.

[14] E. Cuervo, A. Balasubramanian, Dae-ki Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. Maui: making smartphones last longer with code offload. In *Proc. of the 8th Intl conference on Mobile systems, applications, and services*, MobiSys, pages 49–62, New York, USA, 2010. ACM.

[15] L. Deboosere, P. Simoens, J. De Wachter, B. Vankeirsbilck, F. De Turck, B. Dhoedt, and P. Demeester. Grid design for mobile thin client computing. *Future Generation Computer Systems*, 27(6):681 – 693, 2011.

[16] J. Dinan, D. B. Larkins, P. Sadayappan, S. Krishnamoorthy, and J. Nieplocha. Scalable work stealing. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 53:1–53:11, NY, USA, 2009. ACM.

[17] H. T. Dinh, C. Lee, D. Niyato, and P. Wang. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless Communications and Mobile Computing*, 2011.

[18] D. C. Doolan, S. Tabirca, and L. T. Yang. Mobile parallel computing. In *Proc. of the 5th Int'l Symposium on Parallel and Distributed Computing*, pages 161–167, 2006.

[19] D. C. Doolan, S. Tabirca, and L. T. Yang. MMPI a message passing interface for the mobile environment. In *Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia*, MoMM '08, pages 317–321, NY, USA, 2008. ACM.

[20] U. Drolia, R. Martins, Jiaqi Tan, A. Chheda, M. Sanghavi, R. Gandhi, and P. Narasimhan. The case for mobile edge-clouds. In *Ubiquitous Intelligence and Computing, 2013 IEEE 10th International Conference on and 10th International Conference on Autonomic and Trusted Computing (UIC/ATC)*, pages 209–215, Dec 2013.

[21] A. Fahim, A. Mtibaa, and K. A. Harras. Making the case for computational offloading in mobile device clouds. In *Proc. of the 19th Int'l Conference on Mobile Computing & Networking*, pages 203–205, NY, USA, 2013.

[22] N. Fernando, S. W. Loke, and W. Rahayu. Mobile crowd computing with work stealing. In *Proc. of the 15th Int'l Workshop on Mobile Cloud Computing Technologies and Applications (NBiS)*, Sept. 2012.

[23] 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*, volume 120, pages 224–236. Springer Berlin Heidelberg, 2013.

[24] N. Fernando, S. W. Loke, and W. Rahayu. Mobile cloud computing: A survey. *Future Generation Computer Systems*, 29(1):84 – 106, 2013.

[25] N. Fernando, Loke S. W., and W. Rahayu. Dynamic mobile cloud computing: Ad hoc and opportunistic job sharing. In *IEEE Int'l Conference on Utility and Cloud Computing*, pages 281 –286, Dec. 2011.

[26] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. CrowdDB: answering queries with crowdsourcing. In *Proc. of the ACM SIGMOD International Conference on Management of data*, pages 61–72, 2011.

[27] R.K. Ganti, Fan Ye, and Hui Lei. Mobile crowdsensing: current state and future challenges. *Communications Magazine, IEEE*, 49(11):32–39, November 2011.

[28] K. Habak, M. Ammar, K. Harras, and E. Zegura. Femtoclouds: Leveraging mobile devices to provide cloud service at the edge. In *Proceedings of the 8th IEEE International Conference on Cloud Computing*, 2015.

[29] J. Howe. The rise of crowdsourcing. http://www.wired.com/wired/archive/14.06/crowds.html, 2006.

[30] D. Huang, X. Zhang, M Kang, and J. Luo. Mobicloud: Building secure cloud framework for mobile computing and communication. In *Proc. of the 5th IEEE Int'l Symposium on Service Oriented System Engineering (SOSE)*, pages 27 –34, 2010.

[31] G. Huerta-Canepa and D. Lee. A virtual cloud computing provider for mobile devices. In *Proc. of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, pages 61–65, NY, USA, 2010.

[32] V. Jones, A. Halteren, I. Widya, N. Dokovsky, G. Koprinkov, R. Bults, D. Konstantas, and R. Herzog. Mobihealth: Mobile health services based on body area networks. In *M-Health*, Topics in Biomedical Engineering, pages 219–236. Springer US, 2006.

[33] N. Jovanovic and M.A. Bender. Task scheduling in distributed systems by work stealing and mugging - a simulation study. In

Information Technology Interfaces, 2002. ITI 2002. Proceedings of the 24th International Conference on, pages 259 – 264 vol.1, 2002.

[34] R. Kemp, N. Palmer, T. Kielmann, and H. Bal. Cuckoo: A computation offloading framework for smartphones. In *Mobile Computing, Applications, and Services*, volume 76, pages 59–79. Springer Berlin Heidelberg, 2012.

[35] K. Kim, S. Lee, and P. Congdon. On cloud-centric network architecture for multi-dimensional mobility. *SIGCOMM Comput. Commun. Rev.*, 42(4):509–514, September 2012.

[36] M.D. Kristensen. Scavenger: Transparent development of efficient cyber foraging applications. In *Proc. of the IEEE Int'l Conference on Pervasive Computing and Communications (PerCom)*, pages 217 –226, Apr 2010.

[37] K. Kumar, J. Liu, Y. Lu, and B. Bhargava. A survey of computation offloading for mobile systems. *Mobile Networks and Applications*, 18(1):129–140, 2013.

[38] Y. Liu, V. Lehdonvirta, M. Kleppe, T. Alexandrova, H. Kimura, and T. Nakajima. A crowdsourcing based mobile image translation and knowledge sharing service. In *Proceedings of the 9th International Conference on Mobile and Ubiquitous Multimedia*, page 6. ACM, 2010.

[39] S. W. Loke, K. Napier, A. Alali, N. Fernando, and W. Rahayu. Mobile computations with surrounding devices: Proximity sensing and multilayered work stealing. *ACM Trans. Embed. Comput. Syst.*, 14(2):22:1–22:25, February 2015.

[40] W. Lu and D. Gannon. Parallel xml processing by work stealing. In *Proceedings of the 2007 workshop on Service-oriented computing performance: aspects, issues, and approaches*, SOCP '07, pages 31–38, New York, NY, USA, 2007. ACM.

[41] Ya. Luon, C. Aperjis, and B. Huberman. Rankr: A mobile system for crowdsourcing opinions. In *Mobile Computing, Applications, and Services*, volume 95, pages 20–31. Springer Berlin Heidelberg, 2012.

[42] E. E. Marinelli. *Hyrax: Cloud Computing on Mobile Devices using MapReduce*. Carnegie Mellon University, Masters thesis, 2009.

[43] S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang, and A Ghalsasi. Cloud computing the business perspective. *Decision Support Systems*, 51(1):176 – 189, 2011.

[44] S. Mavandadi, S. Dimitrov, S. Feng, F. Yu, U. Sikora, O. Yaglidere, S. Padmanabhan, K. Nielsen, and A. Ozcan. Distributed medical image analysis and diagnosis through crowd-sourced games: a malaria case study. *PloS one*, 7(5):e37245, 2012.

[45] E. Miluzzo, R. Cáceres, and Y. Chen. Vision: Mclouds - computing on clouds of mobile devices. In *Proc. of the 3rd ACM Workshop on Mobile Cloud Computing and Services*, MCS, pages 9–14, NY, USA, 2012.

[46] A. Mtibaa, A. Fahim, K. Harras, and M. Ammar. Towards resource sharing in mobile device clouds: Power balancing across mobile devices. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 51–56, 2013.

[47] D. G. Murray, K. Nilakant, J Crowcroft, and E Yoneki. Task farming in crowd computing. *Mobile Ad Hoc Networking: Cutting Edge Directions, Second Edition*, pages 491–513, 2013.

[48] D. G. Murray, E. Yoneki, J. Crowcroft, and S. Hand. The case for crowd computing. In *Proc. of the 2nd SIGCOMM workshop on Networking, systems, and applications on mobile handhelds*, pages 39–44, 2010.

[49] J. Oomen and L. Aroyo. Crowdsourcing in the cultural heritage domain: Opportunities and challenges. In *Proceedings of the 5th International Conference on Communities and Technologies*, C&T '11, pages 138–149, NY, USA, 2011. ACM.

[50] S. Pandey, W. Voorsluys, S. Niu, A. Khandoker, and R. Buyya. An autonomic cloud environment for hosting ecg data analysis services. *Future Generation Computer Systems*, 28(1):147 – 154, 2012.

[51] R.K. Panta, R. Jana, F. Cheng, Y.R. Chen, and V.A. Vaishampayan. Phoenix: Storage using an autonomous mobile infrastructure. *Parallel and Distributed Systems, IEEE Transactions on*, 24(9):1863–1873, 2013.

[52] K. Parshotam. Crowd computing: a literature review and definition. In *Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference*, pages 121–130. ACM, 2013.

[53] M. Ra, B. Liu, T. La Porta, and R. Govindan. Medusa: a programming framework for crowd-sensing applications. In *Proc. of the 10th int'l conference on Mobile systems, applications, and services*, MobiSys, pages 337–350. ACM, 2012.

[54] J. Ren, Y. Zhang, K. Zhang, and X. Shen. Exploiting mobile crowdsourcing for pervasive cloud services: challenges and solutions. *Communications Magazine, IEEE*, 53(3):98–105, 2015.

[55] J. Rodríguez, C. Mateos, and A. Zunino. Are smartphones really useful for scientific computing? In *Proceedings of the Second International Conference on Advances in New Technologies, Interactive Interfaces and Communicability*, ADNTIIC'11, pages 38–47, Berlin, Heidelberg, 2012. Springer-Verlag.

[56] J. Rodriguez, C. Mateos, and A. Zunino. Energy-efficient job stealing for CPU-intensive processing in mobile devices. *Computing*, 96(2):87–117, 2014.

[57] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE*, 8(4):14 –23, 2009.

[58] C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura. Serendipity: Enabling remote computing among intermittently connected mobile devices. In *Proceedings of the Thirteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, MobiHoc '12, pages 145–154, NY, USA, 2012.

[59] T. Soyata, R. Muraleedharan, C. Funai, M. Kwon, and W. Heinzelman. Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture. In *IEEE Symposium on Computers and Communications*, pages 59–66, 2012.

[60] N. Sultan. Cloud computing: A democratizing force? *International Journal of Information Management*, 33(5):810 – 815, 2013.

[61] The Wi-Fi Alliance. Wi-Fi Direct, 2013.

[62] R. van Nieuwpoort, J. Maassen, G. Wrzesińska, R. F. H. Hofman, C. J. H. Jacobs, T. Kielmann, and H. E. Bal. Ibis: A flexible and efficient java-based grid programming environment: Research articles. *Concurr. Comput. : Pract. Exper.*, 17(7-8):1079–1107, June 2005.

[63] R. V. Van Nieuwpoort, G. Wrzesińska, C. J. H. Jacobs, and H. E. Bal. Satin: A high-level and efficient grid programming model. *ACM Trans. Program. Lang. Syst.*, 32:9:1–9:39, March 2010.

[64] Ti. Yan, V. Kumar, and D. Ganesan. CrowdSearch: exploiting crowds for accurate real-time image search on mobile phones. In *Proc. of the 8th int'l conference on Mobile systems, applications, and services*, MobiSys, pages 77–90. ACM, 2010.

[65] O. F. Zaidan and C. Callison-Burch. Crowdsourcing translation: Professional quality from non-professionals. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 1220–1229. Association for Computational Linguistics, 2011.

Niroshinie Fernando is a Post-doctoral Research Fellow at the Department of Computer Science & Software Engineering at Swinburne University of Technology, Australia. Her research interests include mobile cloud computing, crowdsourcing, social machines, smart cities and IoT. She completed her Ph.D. at La Trobe University, Australia and her B.Sc.(Hons) at University of Colombo, Sri Lanka.

Seng W. Loke is a Reader and Associate Professor at the Department of Computer Science and Information Technology in La Trobe University. He leads the Pervasive Computing Interest Group at La Trobe. He has (co-)authored more than 220 research publications including numerous works on context-aware computing, and mobile and pervasive computing. He has been on the program committee of numerous conferences/-workshops in the area, including Pervasive 2008. He completed his Ph.D. at the University of Melbourne.

Wenny Rahayu is a Professor at the Department of Computer Science and Information Technology, La Trobe University, Australia. Her research areas cover a wide range of advanced databases topics including Spatial and Temporal Databases, XML Databases, Data Warehousing, and Semantic Web and Ontology. To date, she has supervised to completion 10 Ph.D. graduates, and is currently leading a number of collaborative research and industry sponsored projects in the above areas.