2012 15th International Conference on Network-Based Information Systems

Mobile Crowd Computing with Work Stealing

Niroshinie Fernando, Seng W. Loke and W. Rahayu
Department of Computer Science and Computer Engineering
La Trobe University, VIC 3086, Australia
tnfernando@students.latrobe.edu.au, {s.loke,w.rahayu}@latrobe.edu.au

***Abstract*—By pooling together the processing power of mobile devices within a crowd to form a 'mobile cloud', these devices be efficiently utilized to help realize the full potential of mobile computing. However, the dynamic nature of mobile computing makes sharing and coordinating work non-trivial. Although never been used before in the mobile computing domain, the concept of work stealing possesses useful traits such as self adaptiveness, and decentralized nature that can help with these issues. Here we explore this concept of 'work stealing' for crowd computing on an opportunistic network of mobile devices, for both machine and human computation. We also present experimental data and discuss the findings.**

***Keywords*-mobile crowd computing, mobile cloud computing; work stealing; bluetooth**

I. Introduction

Mobile computing can provide a computing tool when and where it is needed irrespective of user movement, thereby supporting location independence. However, the inherent problems of mobile computing such as resource scarcity, finite energy and low connectivity [21] pose problems for most applications. These problems can be addressed by 'sharing' resource intensive work with a resource rich server. However in situations concerning mobile devices, connecting to a remote resource cloud via WiFi or 3G is not feasible because of bandwidth issues, data access fees, and the battery drain [22]. Increasing usage and capabilities of smartphones, combined with the potential of crowd computing [19] can provide a collaborative opportunistic resource pool to solve these problems. We define 'mobile crowd computing' as a local 'mobile resource cloud' comprising of a collection of local nearby mobile devices, utilized to achieve a common goal in a distributed manner.

Work distribution in a mobile environment poses a different set of issues than a typical distributed/grid environment:

1) Less processing power on a mobile device than on s node in distributed processing system.
2) A mobile node has on a finite energy source.
3) A resource pool made up of mobile devices is highly volatile, and hence node availability is inconsistent.
4) In a mobile cloud, the devices will be unknown to each other a priori, unlike in a grid environment where nodes are established and approved beforehand. Therefore a mobile cloud calls for a more opportunistic and ad hoc behavior.
5) A mobile cloud is most likely to be heterogeneous.

Therefore, a mobile resource pool requires a dynamic load balancing method that is decentralized, proactive and self-adaptive instead of the static master-slave work farming. 'Work stealing' [3] is a good candidate for this since it possesses the aforementioned characteristics, and the aim of this paper is to present a work stealing approach to dynamically load balanced *mobile crowd computing*. By mobile crowd computing, we mean machine computation as well as human computation on a crowd (pool) of mobile devices. Here, machine computation refers to work done purely on computers without human intervention, and human computation refers to work done on computers using human expertise.

Although 'work stealing' method has been employed for job scheduling with load balancing in distributed environments such as Cilk ([2], [13]), and Parallel XML processing [17], it has not yet been used in mobile computing domain, as far as we know. The need for dynamic load balancing was demonstrated in [10], where distributed Mandelbrot set generation was done over a set of mobile devices. Mandelbrot set generation is one of the two applications used here as well, but here, we use the work stealing mechanism to efficiently distribute tasks. Work stealing in a 'mobile cloud' would mean connecting opportunistically to unfamiliar devices, while considering the demands of connectivity on the limited battery as well. Therefore, our implementation employs an adjusted version of the traditional work stealing scheme to better suit mobile computing. We show that this mechanism will always give a speedup gain, provided the devices are in no great distance from each other.

II. Motivation

Consider the following scenario as explained in [9]: After a natural disaster such as an earthquake, searching for missing persons is an excruciating task, especially since access to computers and data is limited.

One way of dealing with this is to photograph every person found, gather all images to a central location, and perform search and match operations. However, this is not very realistic considering the limited human and machine resources. Questions that need to be answered in this scenario are, how and who would capture the images?, how would the captured images be collected?, and how would the collected images be processed? Although acquired images could be uploaded into a remote server, connectivity would be a problem, and require a centralized server. Images could be processed locally, but mobile devices are not equipped with enough resources to carry out such operations.

Let us now consider employing a local mobile cloud, where photographs taken by various individuals would constitute the data against which the missing persons will be matched. Relief workers and communities working together could collaboratively 'lend' their mobile resources to a 'local mobile cloud', to do this work.

These kinds of situations call for a system that can function without a priori knowledge of available resources, while load balancing the work among heterogeneous and inconsistent resources.

978-0-7695-4779-4/12 $26.00 © 2012 IEEE
DOI 10.1109/NBiS.2012.122

In [10], we explored processing on the mobile cloud for Mandelbrot set generation, using static work farming. The Mandelbrot image was partitioned, and each partition was assigned to a device, as in Figure 1. This is not efficient because,a)the jobs are not uniform, and b)the participating devices are not of the same processing power. Hence, in a scenario when a stronger Delegator finished its partition before the workers, it has to wait for them. The need for a load balancing method is, therefore evident. Work stealing was chosen since it has shown to be an efficient and scalable load balancing technique for shared and distributed memory systems [7]. Also, it is able to achieve this without a centralized control, and no prior information about the participating devices. As shown in [13] work stealing is efficient even with different processors with dynamically changing speeds.

Mobile crowd computing is applicable for any instance where a group of people are likely to work in close proximity to each other. Although it is not necessary that the group of people are not total strangers, users could be reluctant to share their mobile resources with strangers due to security and privacy reasons. Therefore, a situation consisting of a 'known' community is most likely to succeed. It needs to be noted that we only suggest the users are 'known' to each other, not the devices.

III. Work Stealing in the Mobile Crowd

The concept of Work Stealing on multi processors was first introduced in [4] and [11]. Each process maintains a double ended queue containing the jobs. Each process executes jobs from the head of the queue, and when the queue is empty, attempts to steal jobs from the tail of a queue that belongs to another process. The concept of work stealing in the context of mobile cloud can be explained by Figures 1 and 2 below. Figure 1, illustrates

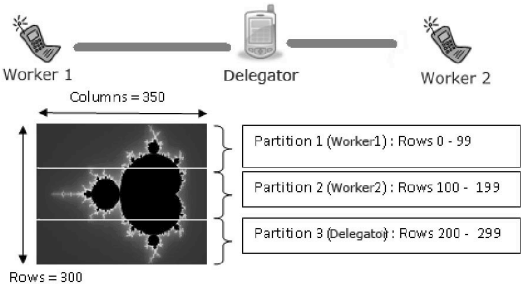

Figure 1. Distributed Mandelbrot set generation using mobile devices

how the Mandelbrot image is initially partitioned into rows and, how these rows are assigned as jobs to the devices in the mobile cloud. Let us say the job pool consisted of nine jobs. As seen in Figure 2, at t_1 time, the delegator has distributed the nine jobs equally among all three participating devices. In this example, we will assume

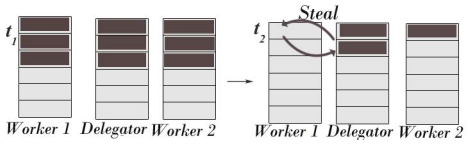

Figure 2. Concept of work stealing in mobile crowd

that worker devices are faster than the delegating device, in the following order; Worker 1>Worker 2>Delegator. After $t_2 - t_1$ time, the delegator has finished one jobs, worker 2 has finished two jobs, and worker 2 all three jobs. Since Worker 2's job queue is now empty, it will try to steal jobs from the delegator, which has two jobs in its job queue.

IV. Algorithm and Implementation

A. Honeybee: Work Stealing implementation

We now give a high level description of our work stealing method for mobile cloud - **Input**: A non-empty list of job parameters. This shall be referred to as the 'job list'. **Output**: An array of computed results corresponding to each job. In the device where the jobs originate from (delegator),

1) Construct job list, and connect to workers.
2) Distribute jobs equally among nodes.
3) Start executing the remaining jobs on the list, while listening for incoming results from workers.
4) If a worker device signals that it wants to steal, examine stealing conditions. If met, assume the role of 'victim' and let the worker 'steal'.
5) If own job list is empty, assume the role of 'thief', select a worker device and try to steal jobs. If stealing attempt is successful, run acquired jobs. If not, select a different worker.
6) Continue this process until the conditions of job completion are met. Upon completion, signal to all workers that the job has been completed and terminate.

In a worker device,

1) Connect to a delegating device, and receive job list.
2) Start executing the job list. Store the results in the completed buffer.
3) If the completed buffer is full, transmit the completed list to the delegator.
4) If the delegator signals it wants to steal, examine the stealing conditions. If met, assume role of 'victim' and let the delegator 'steal'.
5) If own job list is empty, assume the role of 'thief', and try to steal jobs from the delegator. If the stealing attempt is successful, run the acquired jobs. If not, keep trying until delegator sends a termination signal.

Overall, the delegator can steal from 'slow' workers and fast workers can steal from the delegator. In principle, workers could steal from workers, but this requires workers knowing about all other workers, which could affect scalability and have other complications.

B. Constraints and Parameters

1) The jobs are defined by 'Work chunks'. 'Work chunks' define how the total job is partitioned, and determine how many jobs are in the pool.
2) Each device has a 'Steal chunk' specified. Usually, it is advantageous if slower devices have a steal chunk that is greater than faster devices. For example, when a particular device's Steal chunk = 2, a 'thief' can steal upto 2 jobs at a given time.
3) Each device has a unique 'Steal limit' specified. For example, say a device's Steal limit = 1, with a Steal chunk of 5 and has 5 jobs on the list. When it

assumes the role of victim, it will first check the Steal limit, and only release four jobs for stealing. This helps to ensure that a victim will not starve and computations will eventually terminate.

4) The 'Completed jobs buffer' is the number of done jobs stored in a worker, before they are sent back.
5) In our current version the worker devices can not steal from each other. The workers can only steal from the delegator, and the delegator steals from any Worker.
6) The worker that the delegator selects to steal from, is random. Currently, it is the order in which the workers were connected. The delegator will keep on stealing from the first connected worker until it has no more jobs, and move on to the next connected worker.
7) The delegator sends a 'No more jobs to steal' signal if it runs out of jobs. However, a worker will keep on querying the delegator until it receives the 'Termination' signal.
8) The delegating device keeps track of the results it receives/completes by itself, and sends the 'Termination' signal when all jobs are done.

It should be noted that these constraints are not actually constraints on work stealing, but on our particular implementation. Constraint (5) is placed thus for two reasons that are key issues in mobile cloud: minimize the battery drain on a worker, and to minimize the security and privacy issues. Constraint (6) determines our victim selection policy. Existing work on victim selection in non-mobile computing domains such as [16] suggest randomized and round robin selection methods and in [17], a 'Pick-The-Richest' policy has been suggested. However, considering the need to conserve energy with the least amount of communication, we have decided against the 'Pick-The-Richest' policy.

The stealing mechanism is initiated by a device(thief) sending a steal request to a potential victim. When a device receives such a transmission, depending on its available job queue, it can decide to become a victim. The victim then removes a certain number of jobs from its own job list, and transmits them back to the thief. A device can be both a victim and a thief.

V. Experiments

In this section we discuss results from two applications employing work stealing on mobile cloud. Firstly, we discuss our findings from Mandelbrot generation (machine computation). Next we discuss an image acquisition application using human computation.

A. Device specifications

Our test bed contains two Nexus S phones, one Ideos phone and one Nokia X6 phone.

1) Device relative power: Table I provides a comparison of device capabilities for running Honeybee's Mandelbrot implementation. From these results, it can seen that Nexus S is almost the same as Nokia X6, and that Nexus S is roughly 6 times faster than Ideos.

B. Machine Computation: Mandelbrot set generation

We now discuss the results obtained in Mandelbrot set generation. We ran the experiments on a number of configurations; Nexus-Nexus, Nexus-Ideos and Nexus-X6(alternating each device as delegator and worker), Nexus-Nexus-Ideos, and Nexus-Nexus-X6. Table II provides an overview of each test, and its purpose. In the context of Mandelbrot set, 'Work chunks' represent rows of the 300 x 300 Mendelbrot image (see Figure 1), that are grouped together. Therefore, when Work chunk = 10, the total number of jobs in the job pool is 30, with each job consisting of 10 rows of the image. The average speedups for each configuration is displayed in Figure 3. We define a speedup as the time taken to complete job in the cloud divided by the time taken to complete the job on delegating device alone. As can be seen, every configuration has given a speedup greater than 1. Next we display the results obtained for each configuration. The total time spent on the delegating device consists of the calculation time, piconet time, job distribution time, time to set up job pool, time spent on reading results from workers, time spent waiting for results, and time spent on stealing. 'Tc' stands for the time spent on calculation, and 'Tnc' stands for the remaining time. As can be seen, the speedups gained for

Table I
Benchmark results using Mandelbrot algorithm

	200 iterations	**500 iterations**	**1000 iterations**
X6	38,295.00 ms	87,824.00 ms	172,267.00 ms
Nexus	33,194.52 ms	82,480.32 ms	160,413.26 ms
X6/Nexus	1.153654	1.064787	1.073895
	200 iterations	**500 iterations**	**1000 iterations**
Ideos	208,438.06 ms	540,866.00 ms	1,176,953.50 ms
Nexus	33,194.52 ms	82,480.32 ms	160,413.26 ms
Ideos/Nexus	6.2792905	6.5575161	7.337008654

Set	Configuration	Purpose of test
S1	Two devices: Nexus-Nexus	Performance on equal devices
S2	Two devices: Nexus-Ideos, Ideos as delegator	Delegator weaker than worker
S3	Two devices: Nexus-Ideos, Nexus as delegator	Delegator stronger than worker
S4	Two devices: Nexus-X6, Nexus as delegator	Heterogeneous platforms
S5	Three devices: Nexus-Nexus-Ideos, Ideos as delegator	Delegator weaker than two workers
S6	Three devices: Nexus-Nexus-Ideos,Nexus as delegator	Delegator stronger than two workers
S7	Three devices: Nexus-Nexus-X6, Nexus as delegator	Heterogeneous platforms, with two workers

Table II
The Experiment Sets on Mandelbrot set generation

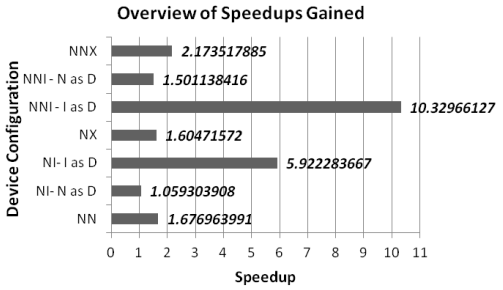

Figure 3. Average Speedups gained in different configurations

various steal chunks only vary slightly, and Tc accounts for the major part of the total time.

1) S1 - Two devices : Nexus S and Nexus S: We first examine the results concerning two identical devices; a Nexus S as the delegating device, and another Nexus S as the worker device. The highest average speedup recorded at 200 iterations was when the steal chunk was 10, at 1.673. For 500 iterations, highest average speedup recorded was when steal chunk was 5, at 1.721.

2) Two devices : Nexus S and Idoes: Experiments with Nexus S and Ideos were twofold; the roles of delegator and worker were assigned interchangeably to both devices.

S2 - Ideos as delegator, Nexus S as worker: An average speedup of 5.92 was recorded for 200 iterations.

S3 - Ideos as worker, Nexus S as delegator: An average speedup of 1.025 at 200 iterations and 1.094 at 500 iterations were recorded.

3) S4 - Two devices : Nexus S and Nokia X6: For this setup, where Nexus S was the delegator and Nokia X6 was thw worker, an average speedup of 1.551 at 200 iterations and 1.659 at 500 iterations were recorded.

4) S5 - Three Devices : Nexus S, Nexus S and Ideos - Ideos as delegating device: Figure 4 shows the results for this setup, with an average speedup of 10.172 at 200 iterations and 10.487 at 500 iterations.

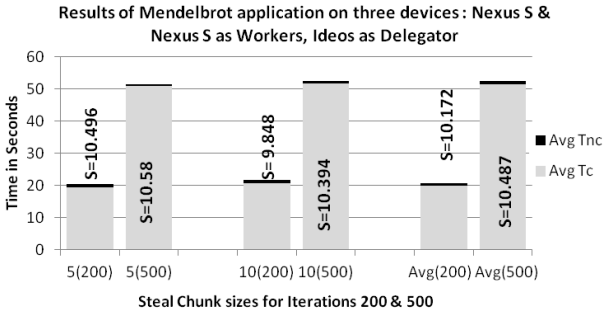

Figure 4. Mandelbrot set generation on Nexus, Nexus and Ideos

5) S6 - Three Devices : Nexus S, Nexus S and Ideos - Nexus as the delegating device: An average speedup of 1.501 was achieved with steal chunk of 5, for 500 iterations, when Nexus S was the Delegating device, and another Nexus S and Ideos were the Worker devices.

6) S7 - Three Devices : Nexus S, Nexus S and Nokia X6 - Nexus S as delegating device: When Nexus S was the Delegator for Workers Nokia X6 and another Nexus S, an average speedup of 2.173 was recorded.

7) Discussion: Here we discuss the findings, regarding primary focus of each experiment set (Table II).

- S1: Two equal Nexus S devices were used to neutralize the effects of heterogeneity and provide an accurate reading of work stealing. This configuration gave an average speedup of 1.697, which is considerably close to the maximum possible speedup of 2 (for this setup). Both devices have spent similar times on job calculation, showing efficient load balancing (Figure 5).
- S2: Work is shared by the weaker Ideos with the stronger Nexus S, giving an average speedup of 5.92. Nexus S is roughly 6.5 times faster than the Ideos (Table I). The effect of a powerful worker is apparent, with performance gain being comparable to the relative speeds between delegator and worker.
- S3:Work is shared by the stronger Nexus S with the weaker Ideos, giving an average speedup of 1.094. Performance gain is negligible, and since the worker is quite slower than delegator, it fails to make an impact. But it should be noted that it still gives a slight gain, even with the overheads of stealing.
- S4: Even across heterogeneous platforms Android and Symbian, work stealing performs well, although the speedup is slightly lower than for two homogeneous devices (set S1).
- S5: A weaker device is sharing work with two stronger devices. This is comparable to experiment S2, where weak Ideos shared work with one stronger device. Comparing the two speedups, it can be seen that by doubling the computational resources, the speedup also increased by almost twofold.
- S6: This is comparable to S3, in that a strong device shares jobs with a weak device. But here, stronger Nexus S shares work with one weak device and one equal device. In S3, adding a relatively weaker device almost had no effect. Here too that seems to be the case, since the average speedup here is 1.5, which is almost the same as S1. In fact in this case, even with slightly more computational resources than in S1, the addition of weaker device has slightly downgraded the speedup.
- S7: From Table I it is evident that Nexus S and Nokia X6 performs almost the same, for Mandelbrot algorithm. Compared to S1, when the number of similar devices was two, this configuration with three devices gives a higher speedup of 2.17, showing a performance increase when scaled up.

Load balancing: Figure 5 shows the time spent on each job on on one execution instance when work chunk was five and run on two Nexus S devices. Since both

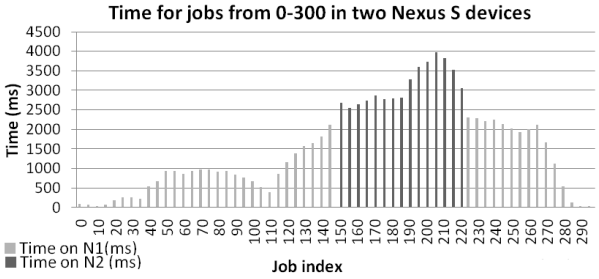

Figure 5. Mendelbrot generation on Nexus breakdown: Time spent on each job

devices are the same, this scenario is useful to show the nature of each job in Mandelbrot generation. Initially, rows 0-149 were assigned to the delegator (N1) and rows 150-299 to the worker (N2). However, rows 225-229 were stolen back by the delegator and more number of jobs have been done by the delegator. Furthermore, it can be seen that each job takes different times to execute, illustrating that jobs are not equal. For example, time to generate rows 0-5 takes 90 ms while rows indexed 145-150 takes 2117 ms. Since jobs are not uniform, time spent on job calculation is the best benchmark for load balancing. When examining the accumulated time spent on jobs by

each device, the calculation times on both devices are almost the same; 46490 ms on the delegator, and 46867 ms on the worker, thus showing that both devices were evenly utilized. This shows the ability of the work stealing approach to deal with non-uniform jobs.

Heterogeneous platforms: The application was implemented on Nokia and Android, and results involving the Nokia device with the Android devices show the successful speedups obtained across heterogeneous platforms.

No prior information on participating devices: One advantage of using the work stealing method is that it requires no prior information about the participating devices. In this implementation, we have not facilitated a meta data exchange between devices, or a benchmarking process to select workers or to assign jobs to them. Furthermore, we have followed a decentralized scheduling approach, and as a whole, the method is self adaptive, which is another key requirement of a mobile cloud, including crowds of mobile devices that are a priori unknown.

Work chunk size (Granularity): Figure 6 shows the results for varying granularity in which the job was partitioned into different chunk sizes. These particular results are for two Nexus S devices executing a 300x300 image for 200 iterations. Although it seems to suggest that greater granularity supports a high speedup in general, the best speedup was received at a medium work chunk size of 10. However, the differences between the speedups are minimal, suggesting that chunk size affects performance, but only slightly, since finer granularity might lead to more even load balancing, but incurs communicating jobs (more stealing).

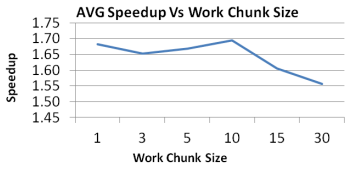

Figure 6. Effect of Granularity on Average Speedup

C. Human Computation: Picture acquisition

Our implementation is based on the following scenario; Consider a setting such as a carnival where many people gather. Typically, visitors are mobile, but stay within a certain boundary for time period. Let us say a person John needs photographic coverage of a parade in the carnival. He would like photographs taken from different perspectives, so he uses his mobile to create a 'job list' that consists of requests for photographs of the parade. In his job requests he specifies parameters such as quality, time interval, deadline to send the photos etc. He then transmits these jobs to nearby mobile 'worker' devices who are willing to do his jobs.

1) Human computation: Instead of machine computation (as in Mendalbrot application), this uses human intervention. When a worker device receives the photo requests, human interaction is needed to comprehend the job parameters and execute the jobs i.e, take photographs. The jobs are grouped by 'interest points' by which, different vantage points for the photographs to be taken are noted. For example, in the parade, these could be given as 'the head of the parade', 'marching band', 'rear of the parade' etc. GPS co-ordinates specifying a locations for photos to be taken can also be used.

2) Work stealing: The job distribution and co-ordination is done according to the work stealing algorithm described in Algorithm section. A worker device that can take photographs quickly, can travel to view the parade from different interest points, is able to finish his/her job list faster than others. Once a worker's job list is exhausted, he/she can send the photographs to the originating device, and has the option of stealing photo jobs from the delegator's job list, and vice versa. Hence our approach of work stealing can be used to load balance jobs done via crowd-sourcing or human computation. For example, a human who can take photographs faster would be on a higher skill level than one who is slower. The faster human can then, finish his/her job queue and steal more jobs from the delegator. In this scenario too, the system has no a priori knowledge of worker capabilities (human skills in photo taking) and work stealing method ensures the system adapts accordingly.

VI. Related Work

Partitioning applications and VM migration are two common methods of offloading and/or sharing work from mobile devices. These include the approach of cyber foraging by Satyanarayan [22]where the work is offloaded a local cloudlet, ensuring better response time. The cyber foraging method is also used by 'Scavenger' framework [15] where jobs are partitioned and distributed via a mobile code approach. CloneCloud [5] uses VM migration to offload through either 3G or WiFi while MAUI [6] uses a combination of VM migration and code partitioning. The [14] 'Cuckoo' framework offloads mobile applications onto local and remote cloud servers, and have re-implemented two existing applications 'eyeDentify' and 'PhotoShoot' to demonstrate the effectiveness of the framework reporting gained speedups and reduced battery consumption. Marinelli [18]'s 'Hyrax' based on Hadoop[1] presents a central server that coordinates data and jobs on connected mobile devices. In [23] Hyrax is further extended to enable searching for and sharing multimedia content on mobile devices. In [20], Palmer *et al*. proposes to use the Ibis grid computing platform to address similar problems in mobile computing, which enables users to integrate their mobile devices onto the grid.The Mobile Message Passing Interface (MMPI) framework [8] is a mobile version of the standard MPI over Bluetooth. Focusing on 'common goals', Huerta-Canepa and Lee [12] present a framework for virtual mobile cloud. Their results do not show a speedup however, although they suggest an energy saving. In [24], different alternatives of implementing a mobile cloud is discussed, and introduce 'Icarus', a mobile storage cloud.

VII. Conclusions and Future Work

Our results with work stealing on mobile devices show that it is a viable method for efficient work distribution in a mobile cloud. We have demonstrated the possibility of a self adaptive and decentralized mobile computation cloud, that is able to obtain performance gains even without prior information about the participating devices. These results are valid for the 'generative' class of applications, where both the machine and human computation applications

[1]http://hadoop.apache.org

shown are 'generative' type, in that the job description is rather small, but the output results in a large amount of data that needs to be transmitted back.

One of the main assumptions in our work so far, is that all the workers are 'well behaved'. In our future work, we hope to address 'rouge workers', to guarantee security and integrity. Redundancy and 'mugging'[13] are possible avenues. Privacy concerns in the mobile cloud are also discussed in [23], who suggest annonymising the source name, or not to record the source name at all. Device participation is an important factor to the success of mobile crowd, and participation depends on the incentives. We hope to include incentive management in our framework in future work, where incentives could be in the form of social contract such as in a group of friends, common goals such as discussed in [12], or monetary as in the case of crowd sourcing done in [1]. Although current experiments have involved only three devices this can further be scaled up to involve many more devices by implementing hierarchical stealing, where workers themselves become delegators. We aim to extend our implementation to use the Amazon cloud as well, since this would provide a comparison between offloading to local versus remote devices.

REFERENCES

[1] Amazon mechanical turk. https://www.mturk.com/.

[2] Robert D. Blumofe, Christopher F. Joerg, Bradley C. Kuszmaul, Charles E. Leiserson, Keith H. Randall, and Yuli Zhou. Cilk: an efficient multithreaded runtime system. *SIGPLAN Not.*, 30:207–216, August 1995.

[3] Robert D. Blumofe and Charles E. Leiserson. Scheduling multithreaded computations by work stealing. *J. ACM*, 46(5):720–748, September 1999.

[4] F. Warren Burton and M. Ronan Sleep. Executing functional programs on a virtual tree of processors. In *Proceedings of the 1981 conference on Functional programming languages and computer architecture*, FPCA '81, pages 187–194, New York, NY, USA, 1981. ACM.

[5] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, EuroSys '11, pages 301–314, New York, NY, USA, 2011. ACM.

[6] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, MobiSys '10, pages 49–62, New York, NY, USA, 2010. ACM.

[7] James Dinan, D. Brian Larkins, P. Sadayappan, Sriram Krishnamoorthy, and Jarek Nieplocha. Scalable work stealing. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 53:1–53:11, New York, NY, USA, 2009. ACM.

[8] Daniel C Doolan, Sabin Tabirca, and Laurence 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, New York, NY, USA, 2008. ACM.

[9] N. Fernando, S. W. Loke, and W. Rahayu. Mobile cloud computing: A survey. *Future Generation Computer Systems*, 2012.

[10] N. Fernando, S.W. Loke, and W. Rahayu. Dynamic mobile cloud computing: Ad hoc and opportunistic job sharing. In *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, pages 281 –286, dec. 2011.

[11] Robert H. Halstead, Jr. Implementation of multilisp: Lisp on a multiprocessor. In *Proceedings of the 1984 ACM Symposium on LISP and functional programming*, LFP '84, pages 9–17, New York, NY, USA, 1984. ACM.

[12] Gonzalo Huerta-Canepa and Dongman Lee. A virtual cloud computing provider for mobile devices. In *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, MCS '10, pages 6:1–6:5, New York, NY, USA, 2010. ACM.

[13] 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.

[14] R Kemp, N Palmer, T Kielmann, and H Bal. Cuckoo: a computation offloading framework for smartphones. In *Proceedings of The Second International Conference on Mobile Computing, Applications, and Services*, MobiCASE '10, 2010.

[15] M.D. Kristensen. Scavenger: Transparent development of efficient cyber foraging applications. In *Proceedings of the IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 217 –226, april 2010.

[16] Vipin Kumar, Ananth Y. Grama, and Vempaty Nageshwara Rao. Scalable load balancing techniques for parallel computers, 1994.

[17] Wei Lu and Dennis 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.

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

[19] Derek G. Murray, Eiko Yoneki, Jon Crowcroft, and Steven Hand. The case for crowd computing. In *Proceedings of the second ACM SIGCOMM workshop on Networking, systems, and applications on mobile handhelds*, MobiHeld '10, pages 39–44, New York, NY, USA, 2010. ACM.

[20] Nicholas Palmer, Roelof Kemp, Thilo Kielmann, and Henri Bal. Ibis for mobility: solving challenges of mobile computing using grid techniques. In *Proceedings of the 10th workshop on Mobile Computing Systems and Applications*, HotMobile '09, pages 17:1–17:6, New York, NY, USA, 2009. ACM.

[21] M. Satyanarayanan. Fundamental challenges in mobile computing. In *Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*, PODC '96, pages 1–7, New York, NY, USA, 1996. ACM.

[22] 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.

[23] Vincent Teo. Mobile cloud computing for data-intensive applications. Technical report, Carnegie Mellon University, 2011.

[24] Paul Woods. Towards a lightweight mobile cloud. Master's thesis, University of Dublin, Trinity College, 2011.