2011 Fourth IEEE International Conference on Utility and Cloud Computing

Dynamic Mobile Cloud Computing: Ad Hoc and Opportunistic Job Sharing

Niroshinie Fernando[1], Seng W. Loke[2] and Wenny Rahayu[3]
Department of Computer Science and Computer Engineering, La Trobe University, Australia
{[1]tnfernando@students., [2]S.Loke@, [3]W.Rahayu@}latrobe.edu.au

***Abstract*—Despite increasing usage of mobile computing, exploiting its full potential is difficult due to problems such as resource sparseness. In this paper, we explore the feasibility of a mobile cloud computing framework to use local resources to solve these problems. The framework aims to determine a priori the usefulness of sharing workload at runtime. The results of experiments conducted in Bluetooth transmission and an initial prototype are also presented. Furthermore, we discuss a preliminary analytical model to determine whether or not a speedup will be possible in offloading.**

***Keywords*-mobile cloud computing; offloading**

I. Introduction

Mobile computing allows the user to be locationally independent by providing a tool when and where it is needed. However, problems arise when trying to support mobility in computing devices: resource sparseness, hazardousness, finite energy source, and low connectivity [18]. Offloading mobile applications to remote clouds such as Amazon EC2 has been discussed [13] where services are provided from the resource rich cloud to the mobile device. But this type of offloading depend on the connection to the remote cloud, and the system fails in low connectivity scenarios [19]. A solution would be to exploit the capabilities of the mobile cloud where a mobile cloud is defined as a 'cloud' of local resources utilized to achieve a common goal in a distributed manner. In this paper, we refer to 'mobile cloud' in a different view than traditional cloud computing. Typically, most of the local cloud resources would be mobile (mobile phones, PDAs, laptops etc) and would be owned by different individuals. A 'mobile cloud computing framework' needs to be dynamic such that it can handle runtime resources and connectivity changes, proactive such that the costs are pre-estimated , opportunistic such that it exploits resources as they are encountered, and cost effective such that its task distribution is based on a cost model with benefits to all participating nodes. Furthermore, such a cloud will not be limited to high end mobile devices, but will be able to cater to low end devices as well. The future computing cloud that surrounds a user will not just be smartphones and PCs, but also computers in shoes, watches, jackets, furniture, cups etc, and such a collection of devices around the user will change as he moves from one environment to another, calling for greater opportunistic behaviour and ad hoc set up, with graceful failing.

Although the concept of offloading to local hardware has been explored in a number of frameworks, these do not satisfy all the requirements of a 'mobile cloud' as defined above. In this paper, we discuss the vision towards such a framework and explore the possibilities based on related work and some preliminary experiments. Challenges in such a framework include mobility, job partitioning, job distribution in the cloud, recognizing a potential cloud device, connectivity options, cost estimation and fault tolerance. Ideally, networking would be accomplished via a common existing short-range protocol such as Bluetooth/WiFi etc, rather than special protocols and we experimentally investigate this possibility in this paper.

II. Motivation

Consider the case of Jane traveling on a train in a foreign country. She needs help planning her trip and wishes to ask from the many locals on the train. To solve the language problem, she starts up a speech recognition and synthesis application in her phone. However, such a program would require much computational power and drain her battery. Fortunately, Jane is able to use her mobile phone to initiate setting up a 'virtual cloud' of local computational resources comprised of the many mobile devices on board.

The aforementioned scenario is only one example demonstrating the need for a mobile cloud computing framework. In wearable computing, two major challenges are to reduce bulkiness and low battery power [1]. This could be solved by offloading the computational jobs to the local 'mobile cloud', while sensors and peripherals facilitate the pervasive experience. In augmented reality, it has been suggested using cloud resources [15] to solve similar problems. Other areas include but are not limited to resource demanding rich media applications, video editing, facial recognition and image search.

We propose comprising the local cloud with other mobile devices, enabling mobility without additional infrastructure. Considering the trends for smart phones, which shows they are getting more powerful each year, a local mobile cloud will be able to provide sufficient resources for intensive mobile apps. Also, since most mobile devices have sensing abilities, a cloud made up of mobile devices will be able to provide the users with context aware services.

In the coming sections, the device that has a job to be completed and initiates the sharing/offloading process will be

978-0-7695-4592-9/11 $26.00 © 2011 IEEE
DOI 10.1109/UCC.2011.45

referred to as 'master' and the other devices doing a share of the job would be referred to as 'slaves/clients'.

III. Proposed Architecture

The main components of the framework can be given as Resource handler, Job handler and Cost handler as in Figure 1. The Resource Handler would be responsible for resource discovery, establishing connections, and exchanging meta data with clients. Meta data includes device details such as CPU, available battery, pricing etc. The Context Manager senses and records contextual information about clients, such as their location, movement, and acceleration which is used by the Resource Monitor to keep track on client nodes. Context information is important to register if new devices are coming in, or if current devices are moving away, resulting in disconnection and fault tolerance mechanisms. Cost handler would need to estimate costs and select suitable client devices based on the device meta data ,user's specific requirements. A micropayment module should also be included to handle the monetary transactions between client devices and master device. Job handler would be in charge of dynamically partitioning the application, creating a job pool and managing a work distribution mechanism. The basic steps are as follows:

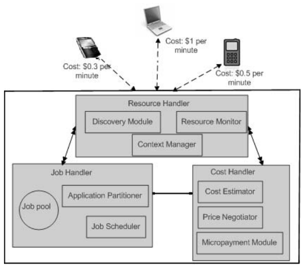


Fig. 1. Main components of a Mobile cloud framework

1) Resource discovery: The master device needs to perform a discovery procedure to search for potential clients within range.
2) Calculate costs: A comparison of potential clients, user priorities, requirements and constraints, is needed to estimate the cost of sharing. Clients can be selected based on this calculation.
3) Distribute jobs: Since client devices will most probably differ in their resources and capabilities, a mechanism to decide which job is assigned to whom is needed. Lessons could be drawn from [9].
4) Run the jobs: Once the jobs have been distributed, the clients would proceed to execute their job/s.
5) Collect the results: When the client devices finish their job/s, results are sent back to the master, and reassembled.
6) Cleanup: Once the job is finished, a method to clean up the client devices of data and/or code that were part of the job is needed to ensure data privacy and security.
7) Handle micropayments: Payments to the client nodes could be processed before or after the completion of the job.

IV. Cost Model, Prototype Implementation and Experimentation

We conducted experiments on Bluetooth to determine its suitability for connectivity on a mobile cloud and the results of these tests were helpful in the construction of our prototype. Our reasons for conducting the preliminary tests on Bluetooth are due to its advantages such as low radiation, low energy cost[6], and wide spread availability as opposed to other protocols such as WiFi [3] and 3G. Also, according to its specifications, future versions will be faster up to 24Mbps and consume less energy.[1] Therefore, when dealing with low end devices Bluetooth would be a better option, although not the only one. Ideally, the system should be able to switch to other protocols depending on the client capabilities and energy situation.

Two phones: Nokia X6 and Nokia 6500, and a PC were used in experiments.These three devices were used since they represent a range (low end mobile, high end mobile, resource rich PC) of devices. The PC used had Microsoft Windows 7 Enterprise with Intel(R) Core(TM) Duo CPU E7300 @ 2.66 GHz 2.67 GHz as processor, 2 GB RAM and Bluecove 2.1.0 on Winsock as Bluetooth driver. To implement Bluetooth communication for J2SE in the PC, we used the third party library BlueCove. According to Nokia's device specifications[2], X6 runs Symbian OS v9.4 with 434 MHz while 6500 runs Nokia OS. Although the CPU details for Nokia 6500 are not available there, other sources such as Softpedia[3] gives it to be 170 MHz.

A. Cost Formulae for Work Distribution

As the primary phase in developing a framework for job sharing, a master-slave system consisting of mobile devices on a piconet was designed using Bluetooth and J2ME. The master node initiates contact and seeks other client nodes advertising 'job sharing' services. Once the master finds client nodes and forms a piconet, it can proceed with sharing/offloading.

1) Sharing jobs: The time for a master node to complete a job is given in equation 1 where T_m is the total time to complete the job in master and T_{pt} is the time to establish the piconet. T_d is the time to partition and distribute jobs to slaves, T_s is the time to complete the master's job and $T_{w\&r}$ is the time the master spends waiting for the slaves. Since computing the master's job and receiving the results are done in separate threads in parallel, the greater value of T_s and $T_{w\&r}$ is used. The pseudo code for the master is given in Listing 1.

$$T_m = T_{pt} + T_d + Max(T_s, T_{w\&r}) \quad (1)$$

The time to complete a job in the slave is given in equation 2 where T_{sl} is the total time to complete the job, T_{cn} is the time to connect to the master, T_{rcv} is the time to receive the job parameters from the master, T_{cp} is the time to complete

[1]https://www.bluetooth.org
[2]http://www.forum.nokia.com/Devices/Device specifications/X6-00/
[3]http://news.softpedia.com/news/Nokia-6500-slide-Review-76693.shtml

```
start master //--(1)--
do device & service discovery
until slave services are found
int numOfdevs = numOfServices + 1
long[] readWaitTimes= new long[numServices];
IF services are found
        FOR each service discovered
                Open a Connection
                Open an outputStream
        END FOR
ELSE
Exit app
END IF //--(2)-- $T_{pt}$ = (2) - (1)
FOR each slave
        generate & distribute parameters
END FOR //--(3)--$T_d$ = (3)-(2)
Start thread ownWork for doing own job
Thread tRead[] = new Thread[numOfServices];
FOREACH tRead
        Open inputStream          //--(4)--
Wait for incoming transmissions from slave_i
        Read result_i
        //--(5)--$T_{w\&r}$[i] = (5)-(4)
END FOREACH
Join all slave threads tRead
and thread doing own job with main thread
//--(6)-$T_s$ = (6)-(3)
//--(7)-$T_{w\&r}$ = add all values in
readWaitTimes array
//--(8)-$T_m$ = (8) - (1)
```

Listing 1. Pseudo code for master

slave's job and T_{sr} is the time to send the completed results back. Listing 2 gives the pseudo code for the slave.

```
start slave//--(1)--
set device as discoverable
start service and wait for connections
open inputStream //--(2)--$T_{cn}$ = (2) - (1)
start receiving job params--(3)--
        WHILE NOT end of inputStream
        read params
        END WHILE
close inputStream
//--(4)--$T_{rcv}$ = (4) - (3)
run job//--(5)--$T_{cp}$ = (5)-(4)
open output stream//--(6)--
Send completed result//--(7)--$T_{sr}$ = (7)-(6)
close connection//--(8)--$T_{sl}$ = (8) - (1)
```

Listing 2. Pseudo code for slave device

$$T_{sl} = T_{cn} + T_{rcv} + T_{cp} + T_{sr} \qquad (2)$$

2) Offloading on a slave: Equation 3 gives the parameters for offloading in the master's perspective. Here, T_{mOff} is the total time to complete the job in the master node, T_{oj} is the time to send job parameters to slave/s and $T_{w\&r}$ is the time the master spends waiting till the slave starts sending results and the time to complete receiving the complete result. The pseudo code for the master is given in Listing 3. The offloaded slave's time is given in equation 4 where T_{sOff} is the total time it takes to complete the job, T_{cn} is the time to connect to the master, T_{cp} is the time to complete the computation and T_{sr} is the time to send the completed results back to the master. The pseudo code for the offloaded slave is the same as for the job sharing slave as given in Listing 2.

```
start master //--(1)--
do device & service discovery until
a slave is found
IF a serivice is found
        open connection
        open outputStream //--(2)--
        //$T_{piconetOff}$ = (2) - (1)
        generate & distribute parameters
        for slave//--(3)--
        //$T_{oj}$ = (3)-(2)
        open InputStream//--(4)---
  read offloaded job results from slave
        close inputStream
        close connection to slave
END IF//--(5)--
//$T_{w\&r}$ = (5) - (4), $T_{mOff}$ = (5) - (1)
```

Listing 3. Pseudo code for master offloading code on slave

$$T_{mOff} = T_{piconetOff} + T_{oj} + T_{w\&r} \qquad (3)$$

$$T_{sOff} = T_{cn} + T_{rcv} + T_{cp} + T_{sr} \qquad (4)$$

3) Analytical model : Offloading to n slaves: We now provide an analytical model generalizing offloading to *n* slaves, showing the conditions under which such offloading results in speedups. We assume that a job is uniformly partitioned such that all slaves get an equal sized job so that a $Job = \sum_{i=1}^{n} J_i$, there is no dependency among each job, and that all *n* slaves in the piconet would be dedicated nodes such that their computational power would be predominantly available for the job. Since $J_1 = J_2 = ... J_i = J_{i+1} ... = J_n$ for n slaves, $Job = nJ$. Therefore, if T_0 is the time to do Job sequentially on master, and if the time it takes the master to run a job J sequentially is T_i, then, $T_0 = nT_i$ (5)
If the computational time for one slave $slave_i$ is Tc_i, then Tc_i would be the computational time to do job J_i. $\therefore T_i \propto Tc_i, \forall i, T_i = k_i Tc_i$ (6)
Where k_i is a constant depending on each slave. Therefore, from (5) and (6), we see that since $T_i = \frac{T_0}{n}$, $\frac{T_0}{n} = k_i Tc_i$ and $\frac{T_0}{Tc_i} = nk_i$. If $nk_i = K_i$ where K_i is a constant, then $\frac{T_0}{Tc_i} = K_i$. $\therefore T_0 = K_i Tc_i$ (7)
If the Speedup is S, and $T_{w\&r}$ is time to wait for and receive all results from slaves, then
$S = \frac{T_0}{T_{master}} = \frac{T_0}{T_m}$
$T_m = T_{pt} + T_{oj} + T_{w\&r}$
$S = \frac{T_0}{T_{pt}+T_{oj}+T_{w\&r}}$ (8)
For *n* devices and a fixed job size of J, T_p and T_{oj} would depend on the bluetooth stack/drivers of the device. Hence, for a given device, let us assume that $T_p + T_{oj}$ would be a constant α. Then, $S = \frac{K_i Tc_i}{\alpha + T_{w\&r}}$ (9)
If t_i is the time the i^{th} thread on master spends waiting and receiving from the i^{th} slave, then the total wait and receive time is equal to the maximum t_i value of the n threads.

This can be represented as: $T_{w\&r} = max\{t_1, t_2, ...t_i..., t_n\}$ (10) If the communication overhead is δ_i, $t_i = Tc_i + Tsr_i + \delta_i$, $T_{w\&r} = max\{(Tc_i + Tsr_i + \delta_i)\}$,
$\therefore S = \frac{K_i Tc_i}{\alpha + max\{(Tc_i + Tsr_i + \delta_i)\}}$ (11)
Since the job size is fixed, Tsr_i solely depends on the device Bluetooth, i.e, the communication cost $Ccomm_i$. Since δ_i is also related to communication overhead, $Tsr_i + \delta_i$ is proportional to $Ccomm_i$. Therefore, if $\beta_i = Tsr_i + \delta_i$, then from (IV-A3), $S = \frac{K_i Tc_i}{\alpha + max\{(Tc_i + \beta_i)\}}$. Here, α depends on the master device, K_i is the ratio of computational times between the master device and the i^{th} slave, Tc_i is the computational time on i^{th} slave, max$\{(Tc_i + \beta_i)\}$ is the maximum value of 1 to n slaves sum of computational time and send result time. Therefore, for $S > 1$, $K_i > \frac{\alpha + max\{(Tc_i + \beta_i)\}}{Tc_i}$. By maximizing the right hand side, since α is related to master's communication cost, the range of α falls between the maximum and minimum values of $MasterComm$. The same applies for Tc_i and β_i which are related to the computation time and communication time of i^th slave respectively. So,
$MasterComm^{min} \leq \alpha \leq MasterComm^{max}$
$SlaveComp_i^{min} \leq Tc_i \leq SlaveComp_i^{max}$
$SlaveComm_i^{min} \leq \beta_i \leq SlaveComm_i^{max}$ (12)
Therefore, the maximum value of $\frac{\alpha}{Tc_i}$, $max[\frac{\alpha}{Tc_i}]$ comes from $\frac{MasterComm^{max}}{SlaveComm_i^{min}}$ and $max[max\{(Tc_i + \beta_i)\}]$ is related to $SlaveComp_i^{max} + SlaveComm_i^{max}$. Therefore, $K_i > \frac{MasterComm^{max} + max\{Cp_i + Cm_i\}}{Cp_i^{min}}$ (13)
Where Cp_i and Cm_i represent $SlaveComp_i$ and $SlaveComm_i$. From (6) and (13), $\frac{T_0}{Tc_i} = K_i$. Since $\frac{T_0}{Tc_i}$ ratio is dependent on the computational capabilities of the devices, if $K_i \propto \frac{CPUslave_i}{CPUmaster}$ then, considering (13), in order to get a speedup greater than one from offloading on *n* devices,
$\frac{CPUslave_i}{CPUmaster} > \frac{MasterComm^{max} + max\{Cp_i + Cm_i\}}{Cp_i^{min}}$ (14)
Based on this derivation, if the ratio between the CPU of the i^{th} slave and the master device is greater than the sum of the communication cost of master and the maximum sum of computation and communication costs out of slaves 1 to *n*, there would definitely be a speedup. The right hand side of the inequality gives the maximum possible value and we have an assumption that the CPU clock rates of the i^{th} slave and master are inversely proportional to their calculation times.

An application of this relation with actual data is given in Section IV-C3.

B. Bluetooth transmission testing

The experiments on Bluetooth were focused on the effect of **message length**, **distance**, **protocol** and **buffer size**.'Buffer size' refers to the size of a message buffer.

1) Message length and Buffer size: Messages of varying lengths were transmitted from Nokia X6 to 6500 for protocols RFCOMM and OBEX for buffer sizes from 256 to 900 B. The trends for all buffer sizes were almost linear, with 500 B giving the best time.

When comparing these two protocols, it could be seen that RFCOMM was faster.

2) Distance: Our tests showed that OBEX has more range than RFCOMM, and as data lengths increase, the maximum range of transmission decreases and so does the consistency.

3) Computational power and heterogeneity: A PC was used to send and receive data to and from Nokia X6 to investigate the effects of computational power and heterogeneity. Figure 2 shows the trend of time against data lengths up to 2 MB in mobile-to-mobile, mobile-to-PC, and PC-to-mobile. Surprisingly, PC to Nokia X6 has the lowest performance,

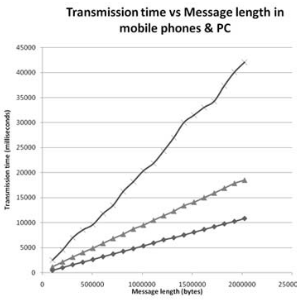


Fig. 2. Bluetooth transmission experiments

while Nokia to PC gives the best time suggesting that although the PC is more powerful than X6, transmission also depends on the receiver's capabilities. It could be that use of third party library to interface with the native Bluetooth stack of the PC (Bluecove) might be adding an overhead. According to the Bluecove speed tests on the web, in some cases mobile to PC is faster, while in other cases, the opposite is true.

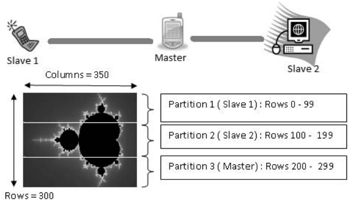
C. Initial Experiments with Distributed Mandelbrot Set Generation

As a first step for a framework for job sharing, we implemented a prototype that partitions a predefined problem and distributes the job among mobile and stationary devices. It was developed using Java, and was executed on the same two mobile phones and the PC that was used in the Bluetooth tests. RFCOMM was used since it had proved to be faster than OBEX in previous experiments.

As a sample application, we implemented a distributed Mandelbrot set generation and compared the results with its sequential implementation on the initiating master device. Although it is not a typical candidate for a mobile application, Mandelbrot generation was selected to study the feasibility and issues in the preliminary tests due to its ease of job partitioning.

1) The Mandelbrot set: The Mandelbrot set is a set of points situated in the complex plane and the boundary of these points forms a fractal. Iterating the formula given in Equation (15) gives this set of points.
$Z_0 = 0, Z_{n+1} = Z_n^2 + C$ (15)
Where C is a constant number on the Complex plane, and Z_n is the current point. As the number of iterations per point increases, the accuracy of the image increases.

2) Parallelizing the Mandelbrot set generation: For a given point, the Mandelbrot iteration feeds the result of the previous step into the next one. Hence, the simplest method of partitioning this algorithm is to assign each device a continuous block of the image. An example of how an image of 300 X 350 pixels is split is given in Figure 3.



Slave 1
Master
Slave 2
Columns = 350
Partition 1 (Slave 1) : Rows 0 - 99
Partition 2 (Slave 2) : Rows 100 - 199
Partition 3 (Master) : Rows 200 - 299
Rows = 300

Fig. 3. Splitting up the Mandelbrot image among three devices

3) Execution results: We tested the application for an image of pixel size 300 X 300 for varying number of iterations from 200 to 5000. The distributed version was tested on two to three devices and the results were compared with the performance of the monolithic version. Figures 4(a,b) shows the results of the Mandelbrot image generation using Nokia 6500 and X6 as the Master.

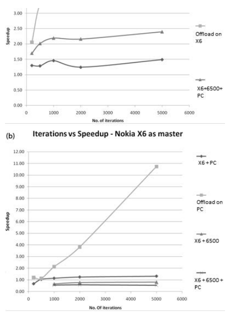


Fig. 4. Iterations vs Speedup (a,b)for different configurations.

Here, Speedup is defined by the following formula in equation (16). $Speedup = T_1/T_m$ (16)

Where T_1 is the execution time of the sequential algorithm on master device, and T_m is the distributed version measured on master device. As can be seen from Figure 4(a), sharing/offloading with Nokia 6500 as master gives speedups from 1.29 up to 3.92. However, the results obtained with Nokia X6 as master show that the distributed version failed to produce a speedup except when it was offloaded and shared with the PC, since X6 is more powerful. When work is shared with a device that is substantially weaker (Nokia 6500), the weak device degrades the overall performance, since the total outcome cannot be given until all the devices complete their work.

Figure 5 shows the processing times in local execution and the shared version in, (A)Nokia 6500 as master and (B)Nokia X6 as master. X axis shows the results of executions for varying Mandelbrot iterations from 500, 1000, 2000 and 5000. Here, 1(A) represent the results of execution for 500 iterations in Mandelbrot generation in which Nokia 6500 was the master and X6 was the slave. Here, the computational time measured on 6500 when it shared the workload with the X6 was only 73% that of its monolithic execution. 1 (B) shows the results for 500 iterations when X6 was master and 6500 was slave. In this case, exeution time on X6 was 84% that of when the Mandelbrot generation was performed on the device (X6) itself. Likewise for results represented in 2,3, and 4. In all cases, though only (A) gave a speedup, it is apparent that the processing time has lessened, suggesting an energy saving [12], given the energy cost for Bluetooth communication is also taken into account.

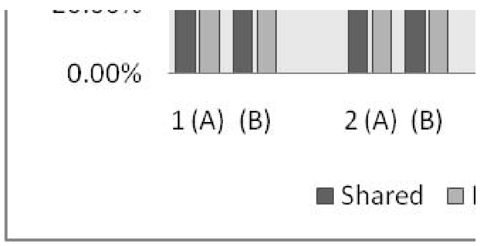


Fig. 5. Processing times Local vs Shared execution where the time for Shared execution is given as a percentage of the Local time.

With regards to Section IV-A3, experimental results above, we identify that the CPU ratio for phones X6 and 6500 as 2.553. When we apply average values from an instance of a series of experiments where the master was 6500 and job was offloaded to X6, the value of $\frac{MasterComm^{max} + max\{Cp_1 + Cm_1\}}{Cp_1^{min}}$ is $\frac{342{,}925.4 + 40{,}668 + 334{,}369}{334{,}369}$,giving the following inequality:

$$2.553 \quad > \quad 2.1472 \tag{1}$$

thereby satisfying the relation in (IV-A3).

V. Related Work

Two common methods relating to offloading and/or sharing work from mobile devices are, partitioning applications and VM migration.

These include the approach of cyber foraging by Satyanarayan [19] for available resources and offload the work through VM synthesis to a local cloudlet. The 'Scavenger' framework [11] also employs cyber foraging.CloneCloud [4] uses VM migration, while MAUI [5] uses a combination of VM migration and code partitioning.Kemp's [10] 'Cuckoo' framework offloads mobile applications onto local and remote cloud servers such as the Amazon EC2.

Marinelli [16]'s 'Hyrax' based on Hadoop[4] presents a central server that coordinates data and jobs on connected mobile devices. The Mobile Message Passing Interface (MMPI) framework [7] is a mobile version of the standard MPI over Bluetooth. Huerta-Canepa and Lee [8] present a framework for virtual mobile cloud focusing on common goals

Enabling mobile clouds with context awareness has been discussed in [14] and [17]. The system proposed in [8] also uses context to identify common goals.

VI. Summary and Future Challenges

Based on our experiments, we surmise that Bluetooth is one of the viable options as the mode of transmission in a mobile cloud. However, our future experiments will not be confined to Bluetooth and would explore other protocols such as WiFi and 3G as well. Also, our experiments show that Bluetooth transmission time is linear against data length and distance and the buffer size plays a substantial role, and RFCOMM outperforms OBEX in terms of speed, but OBEX has more range and computational power alone is not the deciding factor in transmission speed.

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

The results with prototype for distributed Mandelbrot set shows that merely distributing a task regardless of device capabilities will not always give a speedup. In case of equal job sharing, a speedup could only be achieved in cases where the slave node/s were of higher performance than master. Therefore, the device capabilities of the master and potential slaves need to be compared before a job is shared. Tests with offloading show that offloading gives better performance than sharing.

These factors point out the need for a cost model, and we aim to explore 'work stealing' [2] as a way of effective load balancing.

REFERENCES

[1] O. Amft and P. Lukowicz. From backpacks to smartphones: Past, present, and future of wearable computers. *Pervasive Computing, IEEE*, 8(3):8 –13, 2009.

[2] R. D. Blumofe and C. E. Leiserson. Scheduling multithreaded computations by work stealing. *J. ACM*, 46(5):720–748, —1999—. 324234.

[3] S. Cherry. Update: Wifi takes on bluetooth. *Spectrum, IEEE*, 45(8):14, 2008.

[4] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, EuroSys '11, pages 301–314, New York, NY, USA, 2011. ACM.

[5] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, MobiSys '10, pages 49–62, New York, NY, USA, 2010. ACM.

[6] J.-j. Dong and H.-c. Xu. A Distributed Online Test System Based on Bluetooth Technology. In *Proceedings of the Second World Congress on Software Engineering (WCSE)*, volume 1, pages 15 –17, 2010.

[7] 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, New York, NY, USA, 2008. ACM.

[8] G. Huerta-Canepa and D. 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.

[9] K. A. Hummel and H. Meyer. Self-organizing fair job scheduling among mobile devices. In *Self-Adaptive and Self-Organizing Systems Workshops, 2008. SASOW 2008. Second IEEE International Conference on*, pages 230–235, —2008—.

[10] 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.

[11] M. 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.

[12] M. Kristensen and N. Bouvin. Using wi-fi to save energy via p2p remote execution. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on*, pages 123 –128, april 2010.

[13] K. Kumar and Y.-H. Lu. Cloud computing for mobile users: Can offloading computation save energy? *Computer*, 43(4):51 –56, 2010.

[14] H. J. La and S. D. Kim. A conceptual framework for provisioning context-aware mobile cloud services. In *Proceedings of the IEEE 3rd International Conference on Cloud Computing (CLOUD)*, pages 466 –473, 2010.

[15] X. Luo. From augmented reality to augmented computing: A look at cloud-mobile convergence. In *Ubiquitous Virtual Reality, 2009. ISUVR '09. International Symposium on*, pages 29 –32. IEEE, 2009.

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

[17] P. Papakos, L. Capra, and D. S. Rosenblum. Volare: context-aware adaptive cloud service discovery for mobile systems. In *Proceedings of the 9th International Workshop on Adaptive and Reflective Middleware*, ARM '10, pages 32–38, New York, NY, USA, 2010. ACM.

[18] 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.

[19] 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.