A NOVEL SCHEME FOR INCENTIVIZING THE
MOBILE USERS TO SHARE RESOURCES

Name: BHAKTI JAYESH BHADRECHA
Email ID: bjb4484@rit.edu

ADVISOR: Prof. Mohan Kumar
Email ID: mjkvcs@riit.edu
ACKNOWLEDGMENTS

‘A Novel Scheme for Incentivizing the Mobile Users to Share Their resources’ project focuses on optimizing the usage of the mobile resources and creating a distributed computing environment that not only helps to the mobiles users to execute the high resource consumption task efficiently but also helps to incentivize them.

I would like to express my deepest appreciation to Professor Mohan Kumar, who gave me the freedom to explore and at the same time guided me to recover when I faced any obstacle in the implementation of the algorithm. I am deeply grateful to him for those long discussion sessions that helped to brainstorm some amazing ideas and technically helped me to understand the problem statement better. Professor Mohan Kumar is one of the best teachers I have had in my life and I am so thankful to him to encourage me and believe in me, on each and every step towards the implementation of the project. I am also thankful to him for reviewing the project reports and making sure that the reports reflects the hard work done on the project.
# TABLE OF CONTENTS

1. ABSTRACT .......................................................................................................................................... 4

2. INTRODUCTION .................................................................................................................................... 4

3. LITERATURE REVIEW ........................................................................................................................ 5

4. MOTIVATION .......................................................................................................................................... 6

5. SYSTEM ARCHITECTURE ..................................................................................................................... 6
   i. Application Layer .............................................................................................................................. 7
   ii. Distributed Computing Layer ......................................................................................................... 7
   iii. Communication Layer .................................................................................................................... 7

6. IMPLEMENTATION ............................................................................................................................ 9
   i. Introduction ...................................................................................................................................... 10
   ii. Algorithm ......................................................................................................................................... 11
   iii. State Diagram ................................................................................................................................. 12

7. GUI ....................................................................................................................................................... 15

8. RESULTS .............................................................................................................................................. 16

9. CONCLUSION ...................................................................................................................................... 19

10. FUTURE SCOPE .................................................................................................................................. 19

11. REFERENCES ..................................................................................................................................... 19
1. **ABSTRACT**

Today’s smartphone is a powerful computer. It is equipped with a range of sensors and high-bandwidth wireless networking capabilities. The aim of this report is to demonstrate that these smart phones can be used to implement a resources sharing algorithm for shared execution of tasks on remote devices. Inspired by the increasing prevalence of smartphones, this paper evaluates the potential of using these devices to carry out large-scale distributed computations. [2] The other part of this project focuses on incentivizing the mobile users. Providing mobile users a good reason to share their resources is called as incentivizing the mobile users. This is the prime focus of the second part of the project. This paper explains an algorithm that will incentivize the mobile users to share their resources. The results demonstrate that such networks can provide a high degree of parallelism.

2. **INTRODUCTION**

Almost half the phones now sold to consumers are smartphones. Additionally, these smartphones are becoming increasingly more powerful. [2] These resources of smartphones can be used to distribute the high resource consumption tasks and incentivize the mobile users. In order to distribute such tasks among the peer devices, the mobile devices should be connected and form a network. Mobile devices do not have wired capability so they need to be connected using a feature already present in the hardware of the mobile devices. [2] In this case, Wi-Fi is used to connect the devices and share the data among the network of devices.

Now, let’s understand the meaning of Distributed Computing. Distributed computing is a concept where a network is formed in which the data is transferred in a reliable environment. A reliable environment is the one where the delivery of the information from source to destination is guaranteed and also the order of the information is maintained. With the advantages of advanced features in the smart phones, there are also some limitations. The limited amount resources like battery power to execute tasks is one of them. It will be discussed in the paper in detail.

3. **LITERATURE REVIEW**

The battery power of the mobile devices is limited. In order to save the resources and improve the performance of the mobile devices, the computation extensive tasks can be off-loaded to other peer devices/cloud resources. Let’s discuss both the approaches and work done on those approaches.
COSMOS stands for Computation Offloading Services for Mobile Devices. [3] It bridges the gap by providing the offloading computation as a service to the mobile devices. The cloud has resources which could help the mobile devices to execute the tasks fast and efficiently. This will save the resources of the mobile devices. Thus, COSMOS efficiently manages the cloud resources for offloading requests. [3] However, the cloud resources are expensive and they provide per hour rate to the mobile users. Also, the wireless networks like 3G, have low bandwidth which causes long delays for offloading tasks to the cloud. Wi-Fi on the other hand have relatively good bandwidth but limited coverage. There are other issues like, cloud resource management, offloading decisions and allocation of tasks that plays a really important role. Cloud resource focusses on the number of Virtual machines needed to speed up the execution of the offloaded task. Offloading decisions stand for deciding whether a particular task should be offloaded or no. Allocation of tasks generally emphasizes on what virtual machine will process what task. These decisions if not taken properly can impact the performance of the system. These aspects are really important and keeping them in mind, a new approach of eliminating the usage of cloud resources and distributing the workload among the peer devices was introduced. Task farming is another such existing algorithm. The basic idea behind the algorithm is understanding the master-slave architecture, where master is the node that assigns the tasks and slave is the node that receives the tasks and processes it. Once the task is processed, it is sent back to the master. Master node maintains a queue of the tasks and once a slave returns the processed tasks, another task from the master queue is assigned to the slave. The general master slave model is as shown below.

4. MOTIVATION

There are increasing number of real world problems that require processing of very large datasets. In order to achieve the results in a short time and saving resources, it is important to either offload the task or to divide the tasks and distribute them to peer devices. Important reasons so as to why this problem is interesting are:
1. This project emphasizes on implementing a distributed computing framework and use it to solve problems in distributed environment. Few examples of such problems are - Sorting large numbers using merge sort algorithm, measuring frequency of words

2. Processing such large data on a single, fast processing smartphone needs to be highly powerful in terms of its resources. These conditions make the development of the system expensive. Not all can afford such high end system. In order to design a low cost at the same time fault tolerant system, a solution to solve the problem of sorting of large numbers using many networked mobile devices forming a distributed system was introduced.

3. The solution tries to morph a sorting algorithm (merge sort) or in that case any task that needs high computations, with the android devices and also incentivizes the mobile users to share their resources.

5. **SYSTEM ARCHITECTURE**

The system has a Master-Slave architecture. The master node will assign the tasks and the slave node will process the assigned task. The inner components of the master and slave nodes are shown below. It is designed with the aim of being reusable and flexible for further modifications. [2] The solution to a problem keeps evolving. In order to constantly strive for a better solution, the architecture should be open to make any improvements in it.
5.1 Application Layer

The application layer consists of the application that needs to be executed over the distributed wireless network. A task that needs to be executed in a short amount of time, calls for distributed computing on it. In the case of master that application would be a huge file that needs to be sorted and in the case of slave node, it could be a part of the file that needs to be sorted.

5.2 Distributed Computing Layer

The entire logic on the system resides in this layer. The main job of this layer is to decide how the huge task should be distributed over the network in order to reduce the total energy and processing time. It is an important decision as in some cases it makes more sense to compute the task locally because the time taken to process it is less. Distributing it over the network may produce an overhead on the system. On the other hand, some tasks if executed locally can drastically consume the mobile resources and time. Additionally, Distributed Computing Layer is responsible to implement an algorithm that divides the whole task into n number of smaller tasks (depending on the number of the peer
mobile devices available in the network). The details of the algorithm is explained in the section [6].

5.3 Communication Layer

This layer takes care of the reliable transferring of the data between the nodes in the network. In this algorithm, Wi-Fi is used as a medium of connection between the mobile devices. [2] Client socket and server socket are used to carry the data between the master and slave nodes. In this system, there is one master and multiple clients. A multi-threaded application can be a challenge to implement as we need to keep track of the thread socket connections. Synchronization of the code blocks comes into the picture. Synchronization forces a thread to execute the block completely and only then another thread can enter that particular block. Example of sending the data:

```java
bi = new BufferedInputStream(new FileInputStream(file));
bi.read(bytes, 0, byte.length);
OutputStream o = socket.getOutputStream();
o.write(byte, 0, byte.length);
o.flush();
socket.close();
```

The BufferedInputStreamReader can speed up the IO as it reads a large block of the input and buffers it instead of read byte by byte from the network. The size of the buffer plays a very important role in the performance of the reading operation. The FileInputStream makes sure that we read the contents of the file byte by bytes. It supports a read() method that reads the data into a byte array and a close() method that closes FileInputStream after reading all the contents of the file. Example of receiving the data:

```java
File new_file = new File(
        Environment.getExternalStorageDirectory(),
        "test.txt");
byte[] byte = new byte[1024];
InputStream is = new_socket.getInputStream();
FileOutputStream fo = new FileOutputStream(new_file);
BufferedOutputStream bo = new BufferedOutputStream(fo);
    //int bytesRead = is.read(bytes, 0, bytes.length);
    int bread;
while ((bread = is.read(bytes)) > 0) {
    bo.write(byte, 0, bread);
}
    //bos.write(bytes, 0, bytesRead);
bo.close();
socket.close();
```
The BufferedOutputStreamReader can speed up the IO as it write a large block of the data to the network instead of writing it byte by byte. The size of the buffer plays a very important role in the performance of the reading operation. The FileInputStream enables us to write the contents of a file as a stream of bytes.

6. IMPLEMENTATION AND ALGORITHM

The diagram below shows the work flow of the application. Before processing a tasks, points like implementing a logic to divide the tasks and distributing it over the network plays an important role. The section below describes how the slave node (or client) connects with the master node (or server) and how the data transferring between them takes place. The server broadcasts its IP Address to all the clients connected in the network or a group. The client device enters the details of the server and connects to the server. The connection step is done at this point. The data sending and receiving parts are explained with the help of the diagram below.
A wireless application can start a wirelessly distributed job by broadcasting the IP address of the Master node to all the mobile devices in the network.

1. The Master device hosts a network by entering group name and pressing ‘host’ button
   At the slave side, the node enters the group name and IP address of the Master node. The tasks is divided into equal parts depending on the number of nodes available in the network
2. The task is then communicated to the slave node using socket connections and BufferedReader and BufferedWriter class of Java are used to read and write the contents of the file
3. Once the data is received by the node, it starts executing the task
4. In order to make the application incentive, the battery and time consumed by the slave node to execute the task is recorded and the information is transferred back to the master node
5. This whole idea of maintain the record of the battery consumed by one mobile devices in executing the task of another mobile devices helps in incentivizing the mobile users to share their resources

The above two functions are operated in a multi-threaded environment as there are multiple android devices in the network. Also, the Master and Slave nodes are interchangeable. A Master in one network can be a slave in another network.

Once the data is distributed among the peer devices and the processing results are received by the master node, in order to make this approach incentive, a concurrent Hash Map is maintained. This persistent/volatile Hash Map is located globally. A volatile keyword in Java means that, that variable is stored in the main memory. So any changes mad to it is directly shown to other class objects. This Hash Table gives a detail information about how much battery was consumed by the peer device on processing the task of the Master device. For incentivization, this information is necessary.
ALGORITHM

The server in the above diagram encounters a high resource consumption task that needs processing. The flow chart of the actions taking place in the architecture is shown in the diagram below:
Two main parameters of SERVER are:

1. **The threshold** is calculated on the server before dividing the task among the peer nodes. Different sizes of file have pre-determined threshold values. This value is calculated by running the task locally on the server. The threshold value is the minimum amount of time taken by a node to complete the task.

2. **BatteryManager** class of the Android is used to determine the **current battery status** of the server.

Time taken by different file sizes are pre-recorded by doing the same tasks multiple times on the same server. The average is recorded. Also, the battery consumed in doing the same task is recorded. If server has enough battery for process the task then server has two options:

1. Do the task locally
2. Distribute the task among the client nodes in the network
The task is mandatory distributed among the clients if:

1. If the time taken to do the task exceeds the threshold value
2. The battery status of the server is below the critical value

Steps to calculate the current battery status of a node:

```java
// Making sure that the battery is above the critical value
IntentFilter intentFilter = new IntentFilter(Intent.ACTION_BATTERY_CHANGED);
Intent battery_Status = context.registerReceiver(null, intentFilter);

//If battery is not charging and is not full then:
if(battery_Status != BatteryManager.BATTERY_STATUS_CHARGING &&
battery_Status != BatteryManager.BATTERY_STATUS_FULL) {
    int level =
    battery_Status.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);
    int scale =
    battery_Status.getIntExtra(BatteryManager.EXTRA_SCALE, -1);

    int percent = (level*100)/scale;
    return String.valueOf(percent) + "%";

    if((level / (float)scale) < 0.15) {
        return;
    }
}
```

The ACTION_BATTERY_CHANGED intent of the Android system is a broadcast that gives a detailed information about the charging state and the current level of the battery of an android device. The BatteryManager class uses this intent and queries the devices for the information on the charging status and current battery level. The EXTRA for ACTION_BATTERY_CHANGED are EXTRA_LEVEL and EXTRA_SCALE. EXTRA_LEVEL return an integer value that lies between 0 to the maximum battery of the devices i.e. 100%. EXTRA_SCALE return the maximum value of the battery of a device. The current battery value can be calculated as follows:

**Current Battery of a device = (EXTRA_LEVEL/EXTRA_SCALE) * 100 %**

The above code checks if the battery level of the server is above the critical value. The general critical value for android devices is 15%.
To summarize,

**FUNCTIONS OF SERVER:**

- Broadcasting the server details to all the clients in the network
- Maintaining the count of the number of clients in the network
- Count is helped to divide the huge task into small tasks for distributed computing
- Calculating the current battery status of the server
- Threshold is calculated at the server before taking any offloading decision
- A threshold is the time parameter which helps in taking a decision whether a given task should be calculated locally or remotely (on other clients)
- Considering the battery status of the server and the threshold value of the task an offloading decision is made
- If the decision of the server is to offload the task to the clients, then the distributed computing algorithm takes into consideration the battery parameter of the clients as well as the number of clients present in the network
- Battery information of the clients and time taken by them to process a task is also maintained by the server in a table
- This information is used to incentivize the mobile users

**FUNCTIONS OF THE CLIENTS:**

- Once a task is assigned by the server, the battery status of the device is continuously monitored
- If the battery status of the client is not good enough to process the task, it is sent back to the server to re-assign
- Time taken by the client to process the assigned task is recorded
- The time includes the transmission time (time taken to transfer the data from the server and back to the server) as well as the processing time
7. **GUI of the Application**

For a server to host a network, he will need to press the host button. The IP Address of the server will be broadcasted.

For a client to join a network, he will need to enter the IP Address of the server and press the connect button.
8. RESULTS

A task like sending a text message consumes around 300 milliwatts (mW). A Google Nexus mobile device has a battery capacity of 2700 mAh. If it needs to be fully recharged every 32 hours, then the set of tasks performed on it consume on average about 20mW per hour (2.0 x 2700 / 32). These are just estimate. The actual consumption for various tasks differs.

Suppose, the huge resource consumption task is to find out all the occurrences of a words in a given file. The file is divided into mostly equal parts depending upon the number of nodes in a group and the battery of those nodes. These files can be of different sizes. For each file, the average time taken to process it locally is calculated in seconds. Also, the time taken to process the same task in a distributed computing network is recorded too. The below table shows a clear picture of the time taken locally and by the clients in the network. 3 nodes or clients are considered to understand the results.

<table>
<thead>
<tr>
<th>Size of the file</th>
<th>AVG Time taken on Server (secs)</th>
<th>AVG Time taken on 3 nodes (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 KB</td>
<td>3.28</td>
<td>2.12, 1.68, 1.53</td>
</tr>
<tr>
<td>116 KB</td>
<td>4.63</td>
<td>2.32, 3.68, 2.16</td>
</tr>
<tr>
<td>200 KB</td>
<td>7.59</td>
<td>4.63, 3.44, 3.46</td>
</tr>
<tr>
<td>223 KB</td>
<td>10.26</td>
<td>4.98, 5.02, 6.3</td>
</tr>
</tbody>
</table>

Table 1

Let us assume that the server’s battery status is above the critical.

In the first scenario, the maximum of time taken on 3 nodes is:

\[
\text{MAX}(2.12, 1.68, 1.53) = 2.12 \text{ seconds} = t
\]

‘t’ doesn’t include the time to merge the files.
The AVERAGE time taken to merger different file sizes are:

<table>
<thead>
<tr>
<th>Size of the file</th>
<th>AVG Time taken to merge (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 KB</td>
<td>1.05</td>
</tr>
<tr>
<td>116 KB</td>
<td>1.05</td>
</tr>
<tr>
<td>200 KB</td>
<td>1.65</td>
</tr>
<tr>
<td>223 KB</td>
<td>1.66</td>
</tr>
</tbody>
</table>

Table 2

As per the above table, the time taken in to merger a file of size 100 KB is: 1.05 seconds.

Therefore, Total Time = t + 1.05 seconds = 2.12 + 1.05 = 3.17 seconds, which is less than computing the task locally.

NOTE: This time also includes the transmission time to send data from the server to the client and transmission time to send the data back to the server after the task is processed at the client.

In other scenarios, where the file size is 116 KB, the local computation time is 4.63 seconds and the time taken by distributed the workload among the clients in the network is:

MAX(2.32, 3.68, 2.16) = 3.68 seconds = t;

‘t’ doesn’t include the time to merge the files.

As per the above table, the time taken to merge a file size of 116 KB is 1.05 seconds.

Therefore, the Total Time = t + 1.05 = 3.68 + 1.05 = 4.73 seconds, which is greater than the time taken to compute the task locally. So, doing the task locally is less expensive.

So, the decision made by the server is to do the task locally.

NOTE: This time also includes the transmission time to send data from the server to the client and transmission time to send the data back to the server after the task is processed at the client.
Also, in order to incentivize the application, let us again consider table 1.

<table>
<thead>
<tr>
<th>Size of the file</th>
<th>n1</th>
<th>Points of n1</th>
<th>n2</th>
<th>Points of n2</th>
<th>n3</th>
<th>Points of n3</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 KB</td>
<td>2.12</td>
<td>4</td>
<td>1.68</td>
<td>3</td>
<td>1.53</td>
<td>3</td>
</tr>
<tr>
<td>s1 is the server:</td>
<td>116 KB</td>
<td>2.32</td>
<td>2.8</td>
<td>3.68</td>
<td>4.5</td>
<td>2.16</td>
</tr>
<tr>
<td></td>
<td>200 KB</td>
<td>4.63</td>
<td>4</td>
<td>3.44</td>
<td>3</td>
<td>3.46</td>
</tr>
<tr>
<td></td>
<td>223 KB</td>
<td>4.98</td>
<td>3</td>
<td>5.02</td>
<td>3</td>
<td>6.3</td>
</tr>
</tbody>
</table>

Depending the amount of time spent by n1 in doing s1’s task, the points are calculated as follows:

Total time spent by n1, n2 and n3 = 2.12 + 1.68 + 1.53 = 5.33 seconds. 
% of time spent by n1 = ((time spent by n1)* 100)/ (Total time spent) = (2.12 * 100)/ 5.33 = 40
Therefore, number of points in the scale of 1 to 10 for n1 = 40/10 = 4 points. So, s1 owes n1 4 points.

Similarly, the points for n2 and n3 are calculated as shown in the figure above. However, this table holds true for one task when s1 is the server.

Now, in a task where n1 is the server, let us how the algorithm incentivizes the mobile users to share the resources.

Consider the table shown below:

<table>
<thead>
<tr>
<th>Size of the file</th>
<th>s1</th>
<th>Points of s1</th>
<th>n2</th>
<th>Points of n2</th>
<th>n3</th>
<th>Points of n3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100 KB</td>
<td>2.55</td>
<td>4.2</td>
<td>1.7</td>
<td>2.8</td>
<td>1.8</td>
</tr>
<tr>
<td>n1 is the server</td>
<td>116 KB</td>
<td>2.3</td>
<td>2.5</td>
<td>3.5</td>
<td>4.3</td>
<td>2.5</td>
</tr>
<tr>
<td></td>
<td>200 KB</td>
<td>4.5</td>
<td>4.1</td>
<td>3</td>
<td>2.7</td>
<td>3.5</td>
</tr>
<tr>
<td></td>
<td>223 KB</td>
<td>5.3</td>
<td>3.2</td>
<td>5</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

The points are calculated as discussed above. From the above table, n1 owes s1 4.2 points, but we do know that from the first task, s1 owed n1. A table is maintained internally as shown below:

<table>
<thead>
<tr>
<th>Size of the file</th>
<th>s1</th>
<th>n1</th>
<th>n2</th>
<th>n3</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 KB</td>
<td>0.2 points to n1</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>116 KB</td>
<td>0</td>
<td>0.3 points to s1</td>
<td>4.5</td>
<td>2.6</td>
</tr>
<tr>
<td>200 KB</td>
<td>0.1 points to n1</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>223 KB</td>
<td>0.2 points to n1</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
This algorithm works efficiently in most of the cases and it not only helps in distributing the workload and save precious mobile resources, but also incentivizes the mobile users to share their resources among the peer devices. In this way, a distributed computing environment can be created for parallel task execution.

9. CONCLUSION

In this report, the results indicate the potential for Distributed Computing. Smartphones can be used to form a distributed network where the mobile devices can be a master/slave. With the help of the CPU power and battery of those devices, a high resource consumption task can be done in a shorter time and consuming less battery. The algorithm explained in this report is successfully ran on multiple android devices and the time is recorded for each and every device. This time can then be displayed in a table and the results are further used to incentivize the mobile users. The example of a high resource consumption tasks is sorting of a huge file. The algorithm can be used to solve any such problem.

10. FUTURE SCOPE

However, the topics such as server failure, client node failure still remains uncovered. The future scope of this project lies in finding a solution that would back-up the data in the server by creating replicas. If the server node fails to operate, then the back-up become active. In the case of client node failures, every time the client node connects/leaves a network, a notification should be sent to the server and that notification should be carefully analyzed. If the node leaves the network then the data forwarded to that node should be re-distributed among the client nodes in the network.

11. REFERENCES

[1] The Case for Crowd Computing by Derek G. Murray, Eiko Yoneki, Jon Crowcroft, and Steven Hand University of Cambridge Computer Laboratory, Cambridge, United Kingdom {fl}@cl.cam.ac.uk
[2] Wireless Distributed Computing on the Android Platform by Kiran Karra, Thesis submitted to the Faculty of the Virginia Polytechnic Institute and State University in fulfillment of the requirements for the degree of Master of Science in Electrical Engineering
[3] COSMOS: Computation Offloading as a Service for Mobile Devices by Cong Shi, Karim Habak, Pranesh Pandurangan, Mostafa Ammar, Mayur Naik, Ellen Zegura School of Computer Science Georgia Institute of Technology, Atlanta, GA {cshi7, karim.habak, ppandura, ammar, naik, ewz}@cc.gatech.edu