**Mobile Offloading**

**Aakash Kumar**

**(1217137428)**

Arizona State University,

Tempe, USA

akuma219@asu.edu

**Gayathri Alloju**

**(1218539582)**

Arizona State University,

Tempe, USA

galloju@asu.edu

**Prathyusha Kodali**

**(1218986912)**

Arizona State University,

Tempe, USA

pkodali1@asu.edu

**Rohit Vaddi**

**(1217969285)**

Arizona State University,

Tempe, USA

rvaddi1@asu.edu

**ABSTRACT**

**In this project, we have developed a distributed computing infrastructure using mobile phone applications. This project demonstrates mobile offloading between the master and slave applications operated in a distributed environment. This computational infrastructure involves multiple slave devices contributing their computational capacity for the multiplication of matrices. We have also implemented tasks like establishing connection between master and slave devices, monitoring the battery levels and GPS locations of slave devices and comparing the power consumption and execution times of matrix multiplication with distributed computing and without distributed computing.**

***Index Terms* –** Mobile offloading, matrix multiplication, volunteer computing, dispatcher, service discovery, distributed computing.

**INTRODUCTION**

In modern times, the computational need of applications in various fields has increased drastically. Different computing infrastructures have been proposed to meet the high computational requirements of these applications. One of them is distributed computing, which has been shown to be a viable solution designed to address the high computational prerequisites of numerous applications. This approachfollows the assumption that computationally intensive problem can be divided into smaller sub-tasks and distributed across multiple connected devices. When compared to cloud and fog servers, the computational capacity of mobile devices is much lower due to the limitations in resources like battery and storage memory. However, with the recent advancements in technology, researchers have come up with more energy-efficient ways in designing high performance applications using distributed computing infrastructure that reduces the costs of computation, storage and communication. Mobile offloading, is one such technique that allows the smart phones linked through the internet to be included in the distributed environment as computational resources, where in the users of the smart phone devices can opt-in and contribute to the computational network. In this project, we enable real – time volunteer computing by merely offloading the computation in master application to other slave devices.

**PROJECT SETUP**

We have developed our project with the following 5 components – master device, slave device, Google Nearby Connections API, network connection through WiFi or Bluetooth.

*A. Android*

• Android 9.0 (API level 28)

• Android Studio 3.0 or higher

• Permissions: Location, Bluetooth, WiFi State.

• API: Google Nearby Connections API [1] Google Play Services Location [3]

**SYSTEM ARCHITECTURE**

There is a service discovery application in master device which sends request to connect with the accessible phones on the network through Google Nearby Connections API. The API at first attempts to interface through WiFi, at that point attempts to set up a connection over the Bluetooth [1]. The master slave topology of the system followed here is 1-N star geography [2]. A messaging mechanism is implemented to send and get the messages between the master and the slaves. Figure 1 demonstrates the communication between master and slave for the individual tasks. Each message during the communication has relating request code, request status, gadget name and extra data relying upon the sort of the request, for instance, we include matrixes in the payload for matrix multiplication request. Table I sums up the requests for the individual distributed tasks.

**IMPLEMENTATION**

In this project we have developed two android applications to be specific Master and Slave which were implemented with Java utilizing Android Studio. Application improvement can brieﬂy be summed up in the following tasks.

**A. Building up a connection between Master and Slaves**

Master application has a "Find slaves" button on clicking which runs the service discovery application and promotes to each slave in the equivalent Bluetooth and Wi-Fi system of the master. At this point a pop-up appears on the slave application with the decision of accepting or rejecting the connection. If the slave clicks "yes" then a connection is built up and the battery and location i.e, gps data is sent to the master.

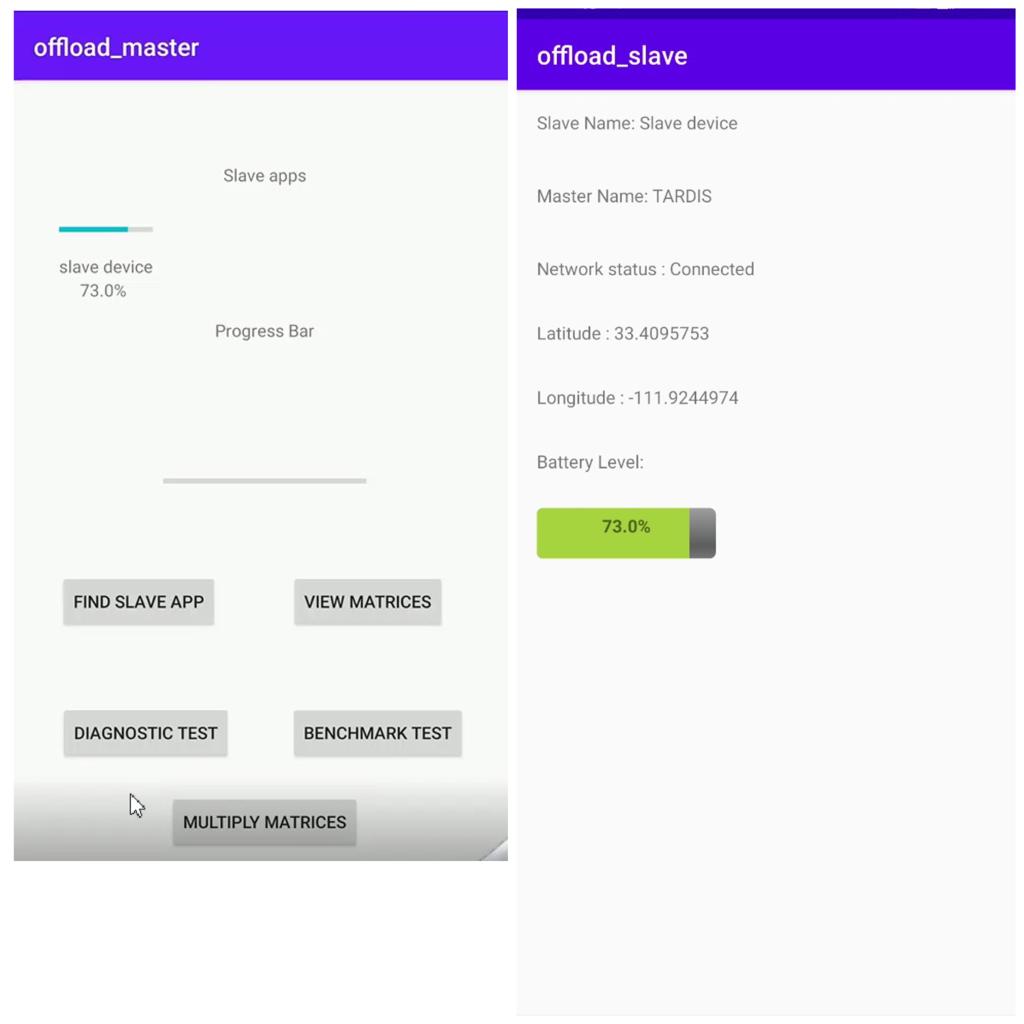


Fig 1.a Connection established

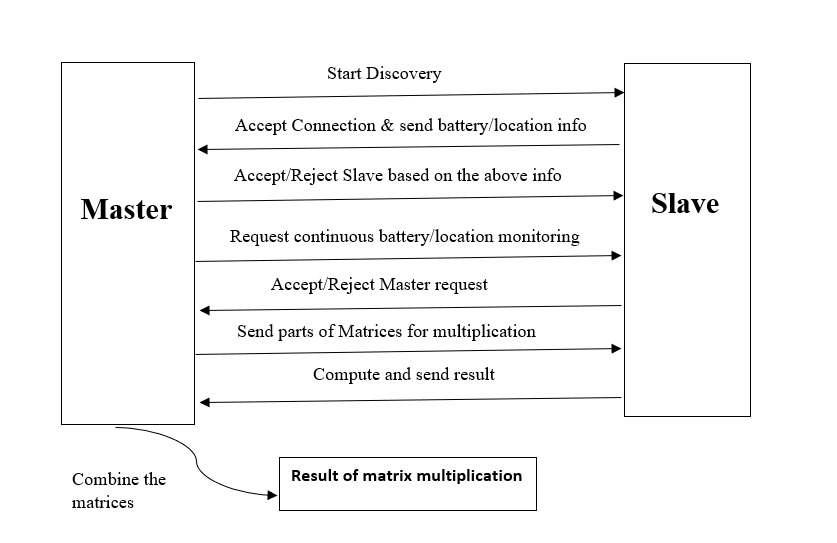


Fig 1 : Mobile Offloading System Architecture

|  |  |  |
| --- | --- | --- |
| Request Code | Description | Payload Example |
| 222 | Request to monitor battery level and location | {  “request\_code”: 222,  “request\_status”: 500,  “name”: ”Gayathri Alloju”,  “battery”: 75,  “latitude”: 33.4095753,  “longitude”: -111.9244974  } |
| 555 | Request for Matrix Multiplication | {  “request\_code”: 555,  “request\_status”:500,  “C”: “[[matrixC]]“  } |

Table 1: Message Payload

**B. Dispatcher Application**

This application ﬁlters out the slaves with whom the connection has been set up with based on least battery level and closeness. The dispatcher show toast that slave does not fulfill least prerequisite since its battery level is not as much as limit of 35%. Each time a payload containing the data about a slave is sent to the master this application is run. Dispatcher assists with optimizing the performance and to not deplete out the battery of slave totally. This feature is integral when a comparative application is run for enormous scope.

**C. Periodic Battery monitoring**

In Master application there is a "Start Monitoring" button. This empowers Master to send a prompt to the slaves to send battery and location data periodically with the user's assent and guarantees the dispatcher to run on updated data, with the goal that it can disconnect any slave if the battery level goes underneath the threshold. The battery level data is stored in "Battery.txt" ﬁle which gets updated at whatever point the slave send updated data regarding its battery status.

**D. Distributed Matrix Multiplication**

We initialize two Matrices randomly on the master application to keep the exhibit of use straightforward. Master can view the matrices A, B and A × B. User is taken to another activity on clicking "View Matrices", where the user can view 'A','B' or 'A×B' by tapping the particular buttons. Master can begin distributed matrix multiplication by clicking "Multiply Matrices" on the principle screen. These matrices are then divided according to the procedure referenced in Algorithm 1. In the first place, when Multiply matrices is invoked, the strategy calls CREATE TASKS. The matrix multiplication is then split into p×p matrices. The estimation of p is set dependent on the network strength. An estimation of p = 10 works useful for most cases. Each one of these tasks are then stored in a Q. In the interim, this procedure also makes a guide Map that tracks what process each task is designated. As of now, this is vacant.

At that point, ALLOCATE TASK is executed for each slave. This expels the ﬁrst element in the queue, produces the slice relating to the indices and sends that to the slave. Even though it is easier to send the whole matrix, this procedure is substantially more efﬁcient since just pieces of the grid are sent which are pertinent to the slaves’ calculation. To streamline this, this procedure is done on the background thread to take into consideration most extreme execution.

**Algorithm** 1 :

procedure CREATE TASKS(p)

Q 🡨[]

Map🡨 {}

for i=0; i<n; i=i+p do

for j=0; j<n; j+=p do

Q 🡨Q + (I,j)

end for

end for

end procedure

procedure ALLOCATE TASK(K)

a,b🡨Q.removefirst()

Map.K🡨(a,b)

send to K:(A[a:a+p,:],B[:,y:y+p])

end procedure

procedure MERGE RESULT(R,K)

a,b 🡨 Map.K

C|x:x + p,y: y+p| 🡨R

ALLOCATE TASK(K)

end procedure

In the event of success, MERGE RESULT is called, which updates the result matrix of the master asynchronously while another task is being assigned to the slave if accessible.

To follow the progress of this method, there is a progress bar on the principle screen of the master which shows the progress of multiplication based on the number of parts of the matrices handled by the slaves.

Request processing on the master's end is finished utilizing queues, which guarantees that the application is fault tolerant as it is easy to follow along and re-route the requests in the event of a failure. A toast indicating the time taken for the multiplication is shown on the master application after the finish. This methodology of distributed multiplication decreases the time taken and conquers any processing limitations of the user when done on enormous grids. Figure 5 shows how the master and slave application look during the calculation.

**E. Estimation of Time taken**

Execution time is estimated for the following scenarios:

1. Matrix Multiplication if done only on the Master
2. If done using the distributed approach with no failure
3. If done using the distributed approach with failure

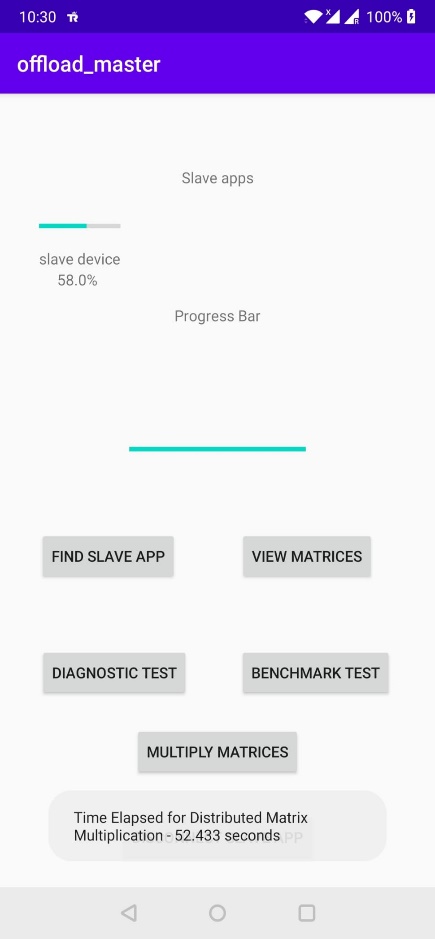


Fig: Time estimation

We have utilized the following formula to calculate the time from the system and the end time was subtracted with the start time to get the time of execution in seconds.

Current Time = System.currentTimeMillis();

Execution Time = (End time - Start time)/1000

The "Benchmark" button on the master shows a toast of the time taken by the master to execute grid multiplication whereas the time taken for distributed matrix multiplication is shown as a toast on the master after completion of the procedure which can be begun by clicking "Multiply Matrices". In spite of the fact that the time taken for the distributed multiplications is higher for lower dimensional matrices, it is much lower compared with the undistributed calculation for higher dimensional grid multiplications which couldn't be shown in this project because of impediment on number of gadgets accessible.

**F. Estimation of Power consumption by nodes**

Estimation of power consumption has become a difficult task with increment in limitations on the data of power consumed by applications in updated version of android API's. There are applications which give a decent estimate of power consumed by each application however it runs on relatively more established forms of android. Thus, we compute the power consumption dependent on the fall in the battery level of the nodes. Although this may not give the absolute power consumption, it gives us an idea of relative power consumption between master and slave nodes. The accompanying equation was utilized for most pessimistic scenario power estimation for multiplication of two 100 x 100 frameworks.

|  |  |  |
| --- | --- | --- |
| Size | Distributed with failure | Distributed without failure |
| 50X50 | 88.63 | 52.4 |
| 100X100 | 162.14 | 136.1 |

Table : Time Comparison

|  |  |  |
| --- | --- | --- |
| Type | Master Power Consumption | Slave Power Consumption |
| Undistributed | 52 mAh | - |
| Distributed | 52 mAh | 82 mAh |

Table : Power Consumption Comparison

A large portion of the power consumption is by the screen and as distributed multiplication tasks take more time for littler dimensions because of communication over the system, the power consumption is more for distributed matrix calculation.

**CONCLUSION**

By implementing this project, we have gained a clear idea of how the connection between master and different slave devices is established and how the computational tasks are offloaded among different slaves using the distributed computing architecture to augment the efficiency of shared infrastructure. This project provided us insights into how mobile offloading technique helps in reducing the computation time taken for matrix multiplication. Additionally, the estimations on the power consumption and execution times on both distributed and centralized environments helped us in understanding the effects they have on the overall performance of the application. This project framework clearly reveals the benefits of using mobile devices in distributed infrastructure for applications that are computationally intensive.

**REFERENCES**

[1] Overview — Nearby Connections API — Google Developers. (n.d.). Retrieved from https://developers.google. com/nearby/connections/overview

[2]Googlearchive.(n.d.).googlearchive/android-nearby. Retrieved from

https://github.com/googlearchive/ android-nearby/tree/master/connections

[3] Get the last known location : Android Developers. (n.d.). Retrieved from https://developer.android.com/ training/location/retrieve-current