CSE 535 Project Report
Mobile Offloading

Anup Kashyap
Department of Computer Science
Arizona State University
Tempe, Arizona
akashy15@asu.edu

Jayashree Natarajan
Department of Computer Engineering
Arizona State University
Tempe, Arizona
jnatara2@asu.edu

Selvaganesh Muthu
Department of Computer Science
Arizona State University
Tempe, Arizona
smuthu4@asu.edu

Sunaada Hebbar Manoor Nagaraja
Department of Computer Science
Arizona State University
Tempe, Arizona
sunaada.hebbar@asu.edu

Abstract *- The goal of this project is to create a distributed computing infrastructure that can be used with mobile devices. An Android application for mobile offloading is developed which uses Bluetooth, Wi-Fi, GPS location and battery information to establish a communication channel between devices. A master-slave architecture is used in this application. The job of the master is to compute the product of two matrices. This matrix multiplication task is divided into sub tasks and shared among the slave devices in the master's proximity. The intermediate results are delivered to the master, which assimilates them and provides the final output after the tasks are distributed and the relevant computations have been completed.*

I. INTRODUCTION

Mobile cloud computing allows users to offload computation activities to other resource-rich mobile devices, lowering battery usage and improving performance. A direct peer-to-peer connection among mobile devices to offload compute chores could be a very promising method for providing a quick mechanism, particularly for time-sensitive and compute intensive jobs. Because it is a heavyweight mechanism requiring substantial power consumption. Using mobile devices to perform calculations has a number of obvious benefits, including greater location flexibility, improved time management, ease of use, and increased productivity. However, it comes at the expense of increased energy use. The loss of power has a direct impact on the battery life of these mobile devices. Distributing the task over numerous devices in a network is one way to solve this challenge. The aggregation of resources, together with the distribution of necessary tasks, considerably reduces each device's power usage. As a result, a matrix multiplication operation that was previously completed by the master device has been split across several devices. As a result, the power consumption of the device is reduced.

II. DESIGN AND IMPLEMENTATION

A single app was developed with two separate modes of operation. This was done in order to avoid code redundancy and to reuse code as much as possible with a common code base. The two modes of operations are “Master” and “Slave”. In the master mode the app starts device discovery and lists all available nearby devices in the discovered devices list on the UI. Then when the user selects a device the app initiates connections with the selected device. When the user runs the app in slave mode, the user can start advertising the device and waits for the any device in master to initiante a connection request. When a connection request is sent by master device, the user with the slave device can either accept or reject the connection request. Once connected, the matrix multiplication operation is started by the master device where the overall operations are divided among all the connected and eligible host devices. The slave devices compute the correspoding results and passses it on to the master device. All the different results are then consolidated into a single matrix and shown on the master device. The flow of the master device is described in figure 1.

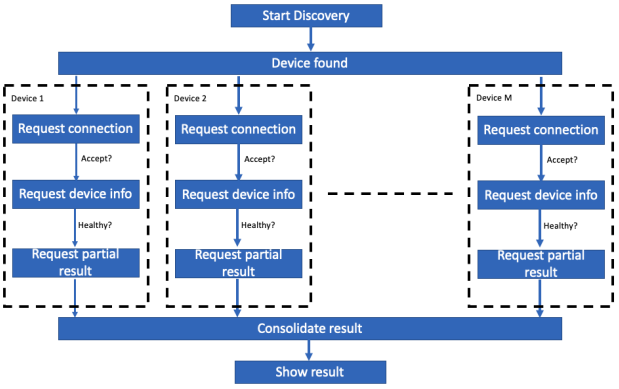

Figure 1: Complete System Flow

The system consists of four major components as follows.

A. Connectivity

A full duplex connection channel had to be established and maintained for exchanging information between devices. Various protocols could be used for this purpose including Bluetooth and Wifi. However instead of directly working with the low level bluetooth or Wifi adapter, we made use of Google's Nearby Connection API[1]. This API abstracts the underlying technologies and provides a platform to discover and connect with other nearby android devices and securely transfer data in multiple configurable modes. However, the Nearby platform still uses technologies such as Bluetooth, Wi-Fi, IP, and audio[2] and makes it easy to discover nearby devices and establish communication with them. We used the "P2P_Cluster"[3] mode of connection which allows communication between multiple devices in both directions.

We set the slave devices as advertisers and discover them using the master device. The Nearby library provides abstract classes for lifecycle management[4] which can be implemented to define actions on various events like new device discovery, device connection or disconnection, device data transfer success or failure, etc. Once the connection is requested from the master device, the slave device asks the user to confirm connection and then successfully establishes connection. In addition, the slave device will also have to give permission for the app to use location information and bluetooth connection since these are critical for establishing connection. The connection session is maintained for further communication between the devices. When a slave device disconnects abruptly due to failure or other issuues the master device gets notified and takes necessary action to handle the failure scenario.

B. Managing devices

In order to distribute tasks across devices and to detect and mitigate device failures, we need to manage and maintain the individual state of each device. We maintain a list of all connected devices with certain properties such as whether the device is still connected, whether the device is busy processing, whether a sub task was assigned to the device , the battery level of the device, location of the device, etc. This information is useful to select slave devices based on customized filters before assigning computation tasks. Furthermore, this info is vital for the fallback mechanism where an unhealthy device is detected and the tasks that were pending on that specific device will be offloaded to another available device. Hence, the master devices continuously monitor the state of slave devices.

The master can choose the devices to connect from the nearby devices. While connecting with the device the master checks the battery status and location information of every device before performing any computation on the device. If the battery is less than 20% master does not perform any computation on the device.

C. Distributed Matrix Multiplication Algorithm

We adopted a naive method to perform the matrix multiplication in a distributed manner. Matrix multiplication basically is performing multiplication between the row elements of the first matrix with the column elements of the second matrix and computing sum of the products repeatedly until the result for each matrix position in the result matrix is computed.

We divided this matrix multiplication into different devices based on the row elements to be computed in the result matrix. If there is an NxN matrix and there are M connected devices, the N rows in the resultant matrix are divided among the M devices. Therefore, each of the M devices will roughly calculate the result for (N/M) rows and send the result to the master device. The master device then reconstructs the final resultant NxN matrix using these partial matrix results rows.

Let the matrices to be multiplied are [a1 a2][a3 a4] and [b1 b2][b3 b4]. Now there are four values to compute in the resultant matrix which is (a1*b1)+(a2*b3) and (a1*b2)+(a2*b4) in the first row and (a3*b1)+(a4*b3) and (a3*b2)+(a4*b4) in the second row. The first row is calculated in the first device and the second row is calculated in the second device. In order to perform the multiplication for the first row the first device will need the first row of the first matrix [a1 a2] and the complete second matrix.

Each healthy slave device connected to the master has to compute N/M rows of the result matrix. We send the indexes of the rows to be computed on the corresponding device. Each device performs the matrix computation in and returns the partial result matrix to the master device. For example, if 4 rows are required to be computed in the result matrix and there are have 2 healthy slave devices, we send indexes 1 and 2 to the first slave device and indexes 3 and 4 to the second slave device for computation. The slave devices return the corresponding rows computed in a single response to the master. The master devices combines all the result rows in the correct order and constructs the final result matrix. This ensures redundant requests are not sent to multiple slave devices. Additionally, this ensures the computation is parallelized and happens asynchronously. The final results is avaialble as soon as all the slave devices return the partial results to the master device.

D. Failure Handling

There are a few scenarios where the slave devices may fail and the master device needs to handle the failure.

1. When a failure occurs on the slave devices before the computation is initiated: In the first step, nearby devices connect to the master device and in the

second step master initiates the distributed computation with the slave devices. So failure can happen at any of these steps. We resolved this issue by maintaining 2 lists, one to have the list of all initially connected devices and another list of active devices containing only healthy devices. The active devices list is obtained by filtering all the connected devices based on various status parameters of every device including battery level, location data, device state, computation task assigned, etc,. Hence the master can select slave devices based on specific parameters and assign computation tasks. Hence, it is always ensured that computation tasks are not assigned to unhealthy or busy slave devices.

2. The second failure scenario can arise when the device fails while the computation task is in progress and master is expecting the partial result from the slave device. In this case the list of rows that was being computed in that particular device must be calculated by some other device. In our system as soon as the compute button is pressed in the master we have a map of whichever devices finished computation and also the rows that are being calculated by every device. Whenever a failure happens the system gets notified about it. The Nearby API provides call back mechanisms for events such as devices getting disconnected or the payload delivery failure[4]. By utilizing these calls we notify the master about the slave device failure, it iterates the list of active devices in the map and gets a device which has already completed its computation and also the list of rows that was being calculated by the failed device will be reassigned to the new active device. In this way master ensures that all the computation tasks are completed even when certain slave devices get disconnected due to device failure or error.

III. TESTING & RESULT

We tested the above system with 4 android devices. One device acted as the master and the other devices acted as the slaves. First we performed the matrix multiplication individually in the master and recorded the time taken.

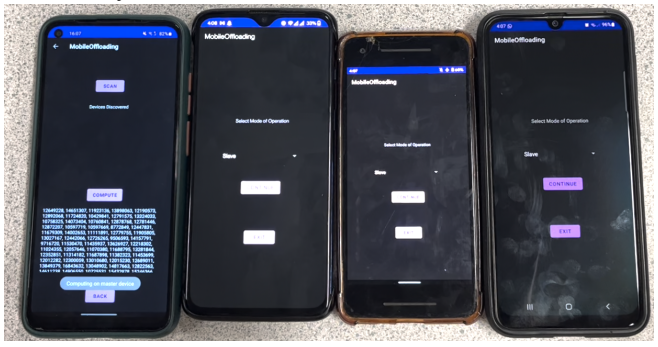

Figure 1: Matrix multiplication in master alone

Then we connected the slave devices and performed the computation in a distributed manner and recorded the time taken.

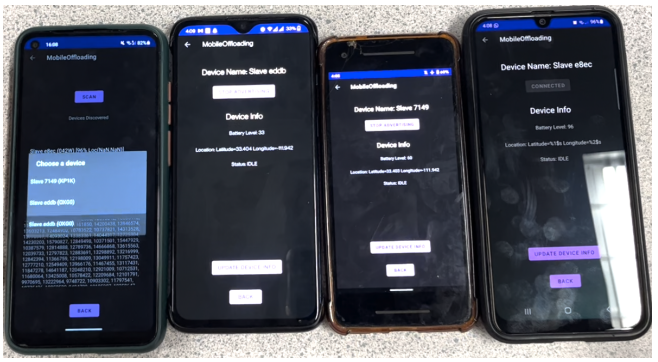

Figure 2: Connecting with Slave devices

Finally, we removed one of the devices and performed the computations again to record the time taken for failure cases.

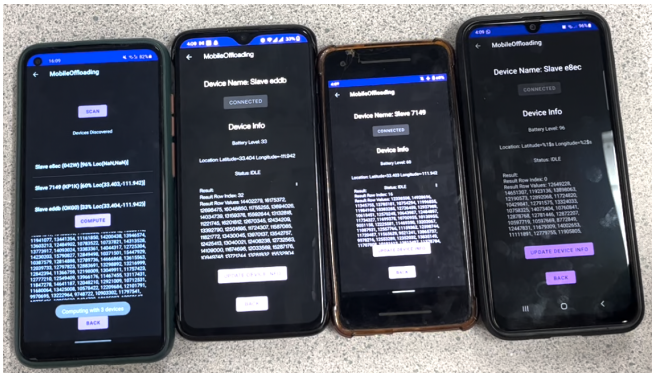

Figure 3: Matrix multiplication in 3 slaves and 1 master

The various execution times for various approaches are listed in Table 1.

No	System State	Time taken (ms)
1	Matrix Multiplication if done only on the Master	235
2	Matrix Multiplication if done using the distributed approach with no failure	3746
3	Matrix Multiplication if done using the distributed approach with failure	4332

Surprisingly, we can see from the above table that the time taken for matrix multiplication in a single master is less than the distributed multiplication. This is because most of the time for distributed multiplication is spent on creating the network connection to multiple devices. It is evident from the second and third point when we connected to 3 devices it took slightly less than when computing with failure. But if we perform matrix multiplication for much larger matrices we will see a significant difference.

We did not notice any differences in the power consumption by the various devices during the various times of computation as we are just trying to calculate the resultant matrix for a 50*50 matrices which were easier to compute.

The power difference from the computation start to end was just 0.

IV. CHALLENGES

Our application's initial and most constraining limitation is the necessity that all mobile devices in the network have Bluetooth, GPS and Wifi-direct capability. Then, keeping in mind the power consumption of the mobile offloading app, a constraint has been placed on slave devices that can be used to offload the computation task. The slave device's battery level must be more than 20% to meet this criterion. If this criteria is not met, the slave node will not be chosen, despite its presence in the network.

V. FUTURE WORK

We have built a system that works in a distributed manner to perform matrix multiplication and it works properly. But there are a few steps that we can take to make the system more efficient.

1. We have built the system around matrix multiplication. We can expand the system in a generic manner such that other operations can be performed easily without much changes.
2. In the case of failure handling, once a device fails we take the list of rows that are being calculated by the device and send it for recalculation with one other device. But there may be several other devices that are idle at that point whose resources are not being used since we perform the calculation on one device only. For example, let us consider a case where there are 3 slaves working on matrix multiplication and each calculating 4 rows each and 2 devices have completed their computations. The remaining device fails. In this case we take the list of rows that was being calculated by the failed device and send it to one of the other devices. Since the other devices are also idle we are not using the resources of the device. We can come with a solution that splits the tasks between the available devices. For instance each device will have to compute 2 rows each from the failed device.

VI. CONCLUSION

The main goal of this project is to convert the limitations of mobile devices (their power consumption) into a strength. Not only is there a significant reduction in the power consumption of individual devices inside the network, but there is also the opportunity to pool resources from all of these devices to complete the desired activity. We do not require high-speed internet connectivity or cloud infrastructure to complete these jobs because they may be completed utilizing only Wifi-direct and GPS capabilities. We are also able to numerically quantify the reduction in power consumption and execution time by offloading the workload.

VII. REFERENCES

[1] Overview | Nearby Connections API | Google Developers
https://developers.google.com/nearby/connections/overview
[2] Nearby | Google Developers
https://developers.google.com/nearby/
[3] Startegies | Nearby Connections API | Google Developers
https://developers.google.com/nearby/connections/strategies
[4] Manage Connections | Nearby Connections API | Google Developers
https://developers.google.com/nearby/connections/android/manage-connections