Mobile Offloading

Ramulu Reddy Challa
Arizona State University
rchally@asu.edu

Manish Aakaram
Arizona State University
maakaram@asu.edu

Setu Durgesh Vinay Annam
Arizona State University
sannam@asu.edu

Nishant Washisth
Arizona State University
nwashist@asu.edu

CSE 535: Mobile Computing
Computing, Informatics, and Decision Systems Engineering
Arizona State University

Under the guidance of
Dr. Ayan Banerjee

Abstract—This project aims to create a distributed computing system with master-worker architecture using mobile phones. The system consists of a master application and worker applications. The connection between the master and worker applications is established using Wi-Fi or Bluetooth (in case Wi-Fi is not available). The master will distribute the work across different workers, which performs the computation to complete the assigned work and send the results back to the master. The master combines the results received from the workers to form the final result. We will be implementing the task of matrix multiplication to show the distributed computation. We also collect important metrics such as worker device battery status every 5 seconds to evaluate the performance of the task when done on master alone and using a distributed approach.

Keywords: Distributed Computation, Matrix Multiplication, Master-Worker Architecture, Performance Analysis.

I. Introduction

In recent times, mobile phones have seen tremendous improvements and enhancements but they still lack the computation powers compared to a high-end laptop or desktop computer. Mobile phones these days are certainly capable of performing complex and hardware demanding tasks such as video processing but as the demand for more complex, hardware and power-hungry tasks increase, the mobiles being a handheld device faces challenges due to slow processor, limited storage, size constraints, data security and lower bandwidth of wireless connections.
Nevertheless, the demand for the highly complex and hardware intensive tasks seems to be constantly growing which is increasing the gap between the capability of a single mobile phone and the highly demanding tasks.
Moreover, mobile phones are being replaced at a very fast rate by their users. According to a study [1], the average life of a mobile phone is around 24 months i.e. on average mobile phone owners replace their mobile phones every 24 months. This leaves a very large amount of unused mobile phones waiting to be disposed of, which could be utilized in the current high computational demanding tasks.
Mobile offloading offers a solution to these problems by combining the computational power of multiple mobile devices to solve the hardware demanding and time-consuming tasks.

Mobile offloading is based on the concept of distributed computing in which a software system is shared among different multiple systems which increases the performance and efficiency of the system. We perform part of Matrix Multiplication as the distributed task on the salve mobile phone applications and combine their results in the master to obtain the final results. We will also take into account the battery levels of the worker mobile phones while performing the distributed computing and will evaluate the performance of the task with and without using the mobile offloading.

II. Architecture

Our master-worker architecture contains a single master and multiple nodes acting as workers.
In this architecture, communication is enabled only between the master and the worker nodes. Worker nodes cannot communicate with each other.

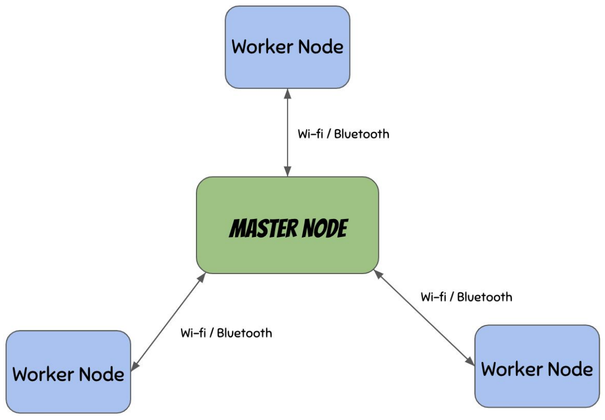


Fig. 1. Master-Worker Architecture

III. Setup

Our single application can enable the device to act as a master node or a worker (not both at the same time). When the user opens the application, they will be given an option to choose if they want to assign work or act as a worker as shown in figure 2.

This application uses Nearby Connections API to enable peer-to-peer communication over local wifi network, and automatically switch to bluetooth in case of wifi unavailability.

Before proceeding, the user will be prompted to explicitly grant the required permissions: ACCCESS_FINE_LOCATION [2].

This application also requires the following permissions [3]:
BLUETOOTH,
BLUETOOTH_ADMIN,
ACCESS_WIFI_STATE,
CHANGE_WIFI_STATE,
ACCESS_FINE_LOCATION,
ACCESS_COARSE_LOCATION.

To support connection between a master node and multiple worker nodes, we use the P2P_CLUSTER [4] connection strategy from the Nearby Connections API.

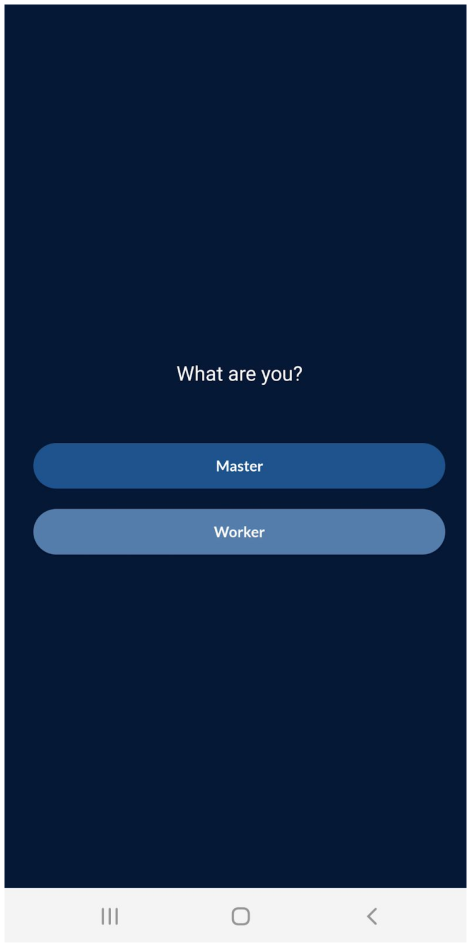


Fig. 2. Home page of the application

Case-(i) As a worker
When the user selects the worker option, they will be redirected to an activity where the client advertisement process is initiated. Applications with the same package name can detect the worker nodes and try to establish a connection. Only the nearby applications can detect if there is worker node advertising.

Case - (ii) As a master
When the user selects the master option, the application will start a client discovery [ref] process and continuously scans for any work advertiser. All the discovered nodes will be displayed on the page, and connection requests will be sent to the respective nodes that chose to be workers.

After a connection is initiated, worker nodes will receive a prompt asking to accept or reject the connection. If rejected, the master node will be informed of this disconnection, and that worker node will be removed from the displayed list of available potential workers. After accepting a connection request from a master, the worker will stop the work advertisement task and the user will be redirected to a different activity where the progress of the work is displayed.

If accepted, the worker node will also send the battery percentage, charging status (if plugged in). The master node will then update the worker connection status from "pending" to "accepted" and show the battery status.

Master can confirm the worker nodes by clicking on the "Done" button as shown in the [Fig]. If there are any workers whose battery level is lower than the threshold value (= 40%), the master node will drop their connections and proceed with the remaining worker nodes. If no worker meets the minimum requirements, the master will close all their connections and application and close the current Activity.

Before the master starts distributing the work, the client discovery task will be ended and the user will be redirected to another activity where the work distribution is visualized in real-time as shown in Fig 3.

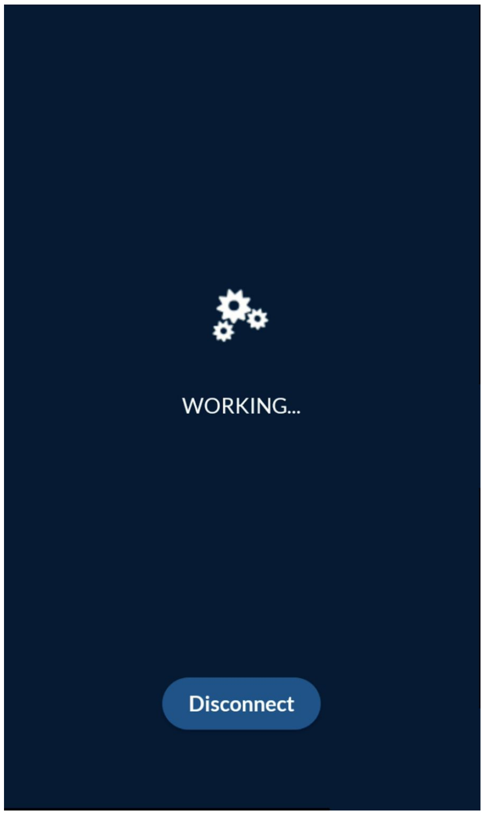


Fig 3. Worker application showing the current status of working

IV. Implementation

Device Monitoring

After accepting a connection from the master node, the worker will start a background thread to send the battery stats (serialized to bytes payload) such as battery percentage (numerical) and the charging status (boolean), and the current location (latitude and longitude) every 5 seconds to the master node. Upon receiving the worker device stats, the master node will update the user interface to show the current battery stats and distance (in meters) between the worker node and master node using the worker's and master's location coordinates obtained from the Google's Fused Location Provider API [5].

Work Partition

The main task of this project is to distribute the work of matrix multiplication to multiple workers and gather the computed results and build the final resulting matrix after multiplication.

Step 1:

Two matrices $A_{(M\times N)}$ and $B_{(N\times P)}$ and created by the master application.

Suppose A x B = C as shown below.

$$\begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \times \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 \\ c_3 & c_4 \end{bmatrix}$$

Here,
$c_1 = a_1 * b_1 + a_2 * b_3$
$c_2 = a_1 * b_2 + a_2 * b_4$
$c_3 = a_3 * b_1 + a_4 * b_3$
$c_4 = a_3 * b_2 + a_4 * b_4$

Step 2:
Our matrix partitioning strategy involves splitting the matrices into multiple arrays in a way that the dot product of two such arrays would result in one of the elements of the final resulting matrix.
For example, if we do a dot product of $[a_1, a_2]$ and $[b_2, b_3]$, we get c_1 in the final matrix.

Applying this strategy, we get m partitions from matrix A and p partitions from matrix B. We combine each row from matrix A and its corresponding column of the same index in matrix B into a single partition.
This would result in the following partitions:

p_1: $[[a_1, a_2], [b_1, b_3]]$
p_2: $[[a_1, a_2], [b_2, b_4]]$
p_3: $[[a_3, a_4], [b_1, b_3]]$
p_4: $[[a_3, a_4], [b_2, b_4]]$

Each such partition is considered a single amount of work that will be sent to a worker. We partitioned the matrices this way so that we are not making partitions of very large size or very small size. If the partition size is very large, and if a worker failed to compute the dot product, then we would have to send all that large data to another work and this would increase the total time of computation. If the partitions are too small, there would be a lot of back-and-forth communication between the master and the worker nodes which could increase total time of computation and drain the battery faster.

By partitioning the matrices this way (combining each row from the first matrix and corresponding column from the second matrix), we avoid such cases during failure or non-failure cases.

Work Distribution
All the available workers are initially put in a heap data structure. Unfinished work is assigned to the next available worker we select from the heap.

The strategy of worker selection
Workers with higher battery level are given higher priority and will be assigned the work immediately if they are available.
We also consider if any worker device is currently plugged-in for charging.

Suppose, if there are two worker devices and the difference in their battery levels is less than 20%, then the device which is being charged is given the higher priority because assigning work to such a device will not drain much battery compared to the non-charging worker device.

If their battery level difference is greater than 20%, then even if the lower battery device is currently being charged, it will not be given the higher priority because such a difference is significant.

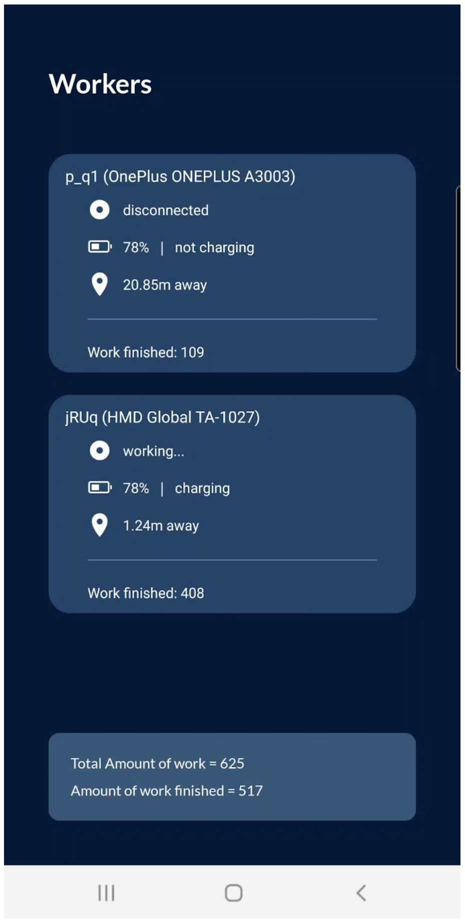
This strategy helps assign more work to the devices which have higher battery level and are plugged-in for charging so that there will not be much battery drain on the lower devices.

After selecting a worker and removing it from the heap, we build payload bytes with the following data: partitionIndex, rows, and cols.

The worker will be notified of any received payload and it will deserialize the bytes to Object and extract the rows and cols. After extracting the rows and cols arrays, we perform a dot product on them and send the resulting value along with the received partitionIndex back to the master node as bytes.

When a payload is received from a worker node, the master node will read in the partitionIndex and the result value after deserializing the bytes. It will then mark the work with this partitionIndex as finished and add the worker back to the heap. When a worker is pushed into the heap, the heap structure will be adjusted according to our worker selection strategy.

We keep track of which work is finished to make sure that all the work is finished by the available workers. When all the work is finished, the connected workers will receive a farewell message notifying them that there is no more work to assign and all the connections will be closed.


Workers

p_q1 (OnePlus ONEPLUS A3003)
disconnected
78% | not charging
20.85m away
Work finished: 109

jRUq (HMD Global TA-1027)
working...
78% | charging
1.24m away
Work finished: 408

Total Amount of work = 625
Amount of work finished = 517

Fig 4. Device (p_q1 OnePlus A3003) is disconnected and it's current status is shown as "disconnected" in the master application

Failure Recovery

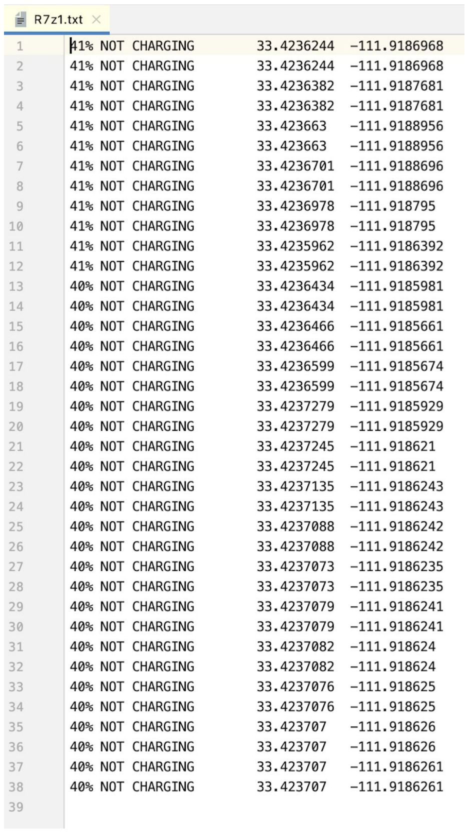
There are cases where a worker may get abruptly disconnected from the master node or the user explicitly chooses to close the connection. In either case, the master will be notified of the worker's disconnection and the work distribution service will remove the worker from the heap data structure. If a worker is disconnected, the work previously assigned to it will now be assigned to the next available worker.

When a worker is disconnected, the battery status monitoring task will be finished and we will also update the worker status to "disconnected" to show to the user as shown in Fig 4.

V. Performance Analysis

Performance of our distributed approach of matrix multiplication is calculated using the following metrics:

- amount of power consumed before all the work is finished by the master and the workers if done on master alone and using the worker nodes.
- the time taken to compute the multiplication on the master node alone and with using the distributed strategy.



R7z1.txt

```
41% NOT CHARGING        33.4236244  -111.9186968
41% NOT CHARGING        33.4236244  -111.9186968
41% NOT CHARGING        33.4236382  -111.9187681
41% NOT CHARGING        33.4236382  -111.9187681
41% NOT CHARGING        33.423663   -111.9188956
41% NOT CHARGING        33.423663   -111.9188956
41% NOT CHARGING        33.4236701  -111.9188696
41% NOT CHARGING        33.4236701  -111.9188696
41% NOT CHARGING        33.4236978  -111.918795
41% NOT CHARGING        33.4236978  -111.918795
41% NOT CHARGING        33.4235962  -111.9186392
41% NOT CHARGING        33.4235962  -111.9186392
40% NOT CHARGING        33.4236434  -111.9185981
40% NOT CHARGING        33.4236434  -111.9185981
40% NOT CHARGING        33.4236466  -111.9185661
40% NOT CHARGING        33.4236466  -111.9185661
40% NOT CHARGING        33.4236599  -111.9185674
40% NOT CHARGING        33.4236599  -111.9185674
40% NOT CHARGING        33.4237279  -111.9185929
40% NOT CHARGING        33.4237279  -111.9185929
40% NOT CHARGING        33.4237245  -111.918621
40% NOT CHARGING        33.4237245  -111.918621
40% NOT CHARGING        33.4237135  -111.9186243
40% NOT CHARGING        33.4237135  -111.9186243
40% NOT CHARGING        33.4237088  -111.9186242
40% NOT CHARGING        33.4237088  -111.9186242
40% NOT CHARGING        33.4237073  -111.9186235
40% NOT CHARGING        33.4237073  -111.9186235
40% NOT CHARGING        33.4237079  -111.9186241
40% NOT CHARGING        33.4237079  -111.9186241
40% NOT CHARGING        33.4237082  -111.918624
40% NOT CHARGING        33.4237082  -111.918624
40% NOT CHARGING        33.4237076  -111.918625
40% NOT CHARGING        33.4237076  -111.918625
40% NOT CHARGING        33.423707   -111.918626
40% NOT CHARGING        33.423707   -111.918626
40% NOT CHARGING        33.423707   -111.9186261
40% NOT CHARGING        33.423707   -111.9186261
```

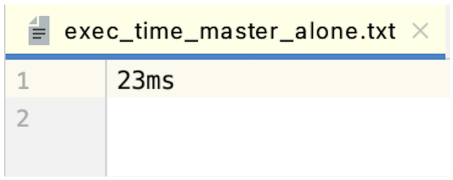
Fig 5. Battery level, charging status, location coordinates of worker node (endpointID = r7z1 same as the file name)

Whenever the master receives the device status from the worker nodes, it will write them to a text file whose name is the same as the endpoint ID of the device in our network.

As shown in the Fig 5, we are storing the battery percentage, charging status, and the location coordinates of the worker device in a text file every 5 seconds if there is no disconnection. When a worker's device is disconnected, we do not append anything to this text file.

From these data, we can calculate the amount of a worker's battery level dropped by the end of all the work.

We also store the time taken (in milliseconds) to finish the matrix multiplication on the master node alone in a text file named "exec_time_master_alone.txt"

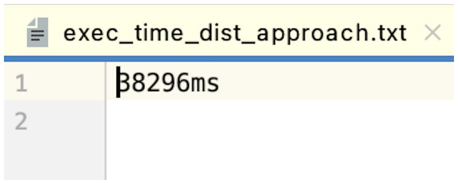


exec_time_master_alone.txt

```
23ms
```

Fig 6. Execution time of two 150x150 matrices multiplication without distributed computation

Before starting the matrix multiplication task, both the matrices are stored in files named “matrix1.txt” and “matrix2.txt” respectively. At the end of the distributed computing, we write the time taken (in milliseconds) to finish the multiplication of the same matrices in a text file named "exec_time_dist_approach.txt"



exec_time_dist_approach.txt

```
38296ms
```

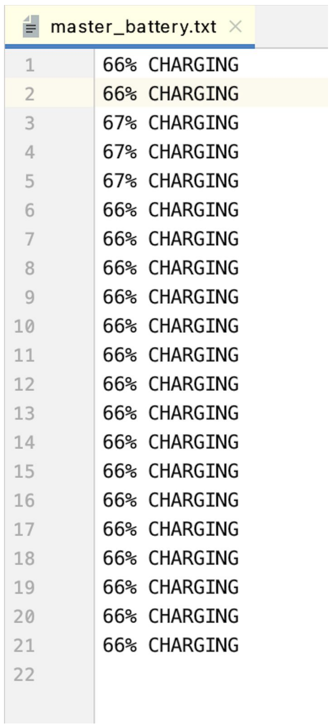
Fig 7. Execution time of two 150x150 matrices multiplication with distributed computation

When all the work is finished, that is when we compute every element of the final matrix, we write this final resulting matrix to a file named “res_matrix.txt”.

Execution time and power consumption:

Strategy	Execution Time (ms)	Power consumption (Master)	Power consumption (Worker1, Worker2)
No distributed computation	23	1%	-
Distributed computation (No failure)	88296	0%	1%, 0%
Distributed computation (With failure)	183024	1%	2%, -

Comparing execution times and power consumptions of distributed and non-distributed (master alone) approach of multiplying two 150x150 matrices.
Column 1: scenario in which we tested the task
Column 2: execution time in milliseconds
Column 3: Power consumption on master
Column 4: Power consumption of two workers. If a worker is disconnected or irrelevant for a scenario, we represent it with a "-"



```
66% CHARGING
66% CHARGING
67% CHARGING
67% CHARGING
67% CHARGING
66% CHARGING
66% CHARGING
66% CHARGING
66% CHARGING
66% CHARGING
66% CHARGING
66% CHARGING
66% CHARGING
66% CHARGING
66% CHARGING
66% CHARGING
66% CHARGING
66% CHARGING
66% CHARGING
66% CHARGING
66% CHARGING
```

Fig 8. Battery status of the master device when performed matrix multiplication without distributed approach

Observations:
When we performed the matrix multiplication of two 150x150 matrices, there was a ~1% of power consumption on master [Fig 8] but it finished the execution in about 23 milliseconds. There was a drop in the battery level but since the device was being charged, it got back up by 1%. But the computations are heavy enough to drag the percentage down by 1% again. That's why we see the battery level transition as 66% → 67% → 66%

Distributed computation incurred more communication cost as we can see that the execution times of distributed approach are higher than that of non-distributed approach. This is because of the partition size of the matrices. With matrices split into many partitions, there is too much back-and-forth communication between the master and the worker but this would be efficient in the cases of heavy failures.

In scenario 3, when we disconnected the Worker2, it took more time to finish the task because all the work has been assigned to a single worker. This also reduced the battery percentage by 2%.

VI. Future Work

Since not every device has the same battery capacity, some devices tend to have a faster battery drain compared to the higher capacities ones. Therefore, it is advisable to also consider the total battery capacity to select the next available worker so that the devices which have higher battery level percentage at the moment but lower capacity will not be assigned more work. This could potentially reduce the huge battery drains on the lower end mobile devices. We could improve this further by also considering the power to performance ratio of the mobile devices. Choosing a correct and adaptable partition size of the matrices could hugely improve the overall performance of the distributed systems.

VII. Conclusion

In the project, we successfully developed a distributed computing system called Mobile Offloading to perform the matrix multiplication and analyzed the results with and without offloading. We implemented the master-worker architecture to recover from the failure of any worker node and be able to finish the work with the rest of the workers. We learned that partition size of the work could hugely affect the overall execution time of the task but is ideal in the cases of heavy failures to avoid reassignments of large amounts of work.

VIII. References

[1] A. Ng, “Smartphone users are waiting longer before upgrading - here's why,” *CNBC*, 17-May-2019. [Online]. Available: https://www.cnbc.com/2019/05/17/smartphone-users-are-waiting-longer-before-upgrading-heres-why.html.

[2] *Google*. [Online]. Available: https://developers.google.com/maps/documentation/android-sdk/location. [Accessed: 11-Dec-2020].

[3] “Get started | Nearby Connections API | Google Developers,” *Google*. [Online]. Available: https://developers.google.com/nearby/connections/android/get-started. [Accessed: 11-Dec-2020].

[4] “Strategies | Nearby Connections API | Google Developers,” *Google*. [Online]. Available: https://developers.google.com/nearby/connections/strategies. [Accessed: 11-Dec-2020].

[5] “Request location updates : Android Developers,” *Android Developers*. [Online]. Available: https://developer.android.com/training/location/request-updates. [Accessed: 11-Dec-2020].