Mobile Offloading Application

Anvesh Kumar Voona
Arizona State University,
Tempe, USA
avoona@asu.edu

Midhun K Gopalakrishnan
Arizona State University,
Tempe, USA
mkgopala@asu.edu

Sainath Latkar
Arizona State University,
Tempe, USA
slatkar1@asu.edu

Yuchen Ge
Arizona State University,
Tempe, USA
yuchenge@asu.edu

***Abstract*—In this project, we create a distributed computation infrastructure using mobile phones for solving a complex computational problem of matrix multiplication. The distributed computation infrastructure works in master-slave mode meaning one of the device acts as a master and orchestrates the process. Other devices acting as slaves or workers, get computational task from master, compute matrix multiplication and send back the result to the master. Master node aggregates all the results sent by the distributed devices and generate a resultant matrix. We use Wi-Fi and/or bluetooth connectivity for sharing the data across distributed network of mobile phones which are in the vicinity of the master node. We also compare the performance of a single device doing matrix multiplication against performance of the same problem being solved on a distributed infrastructure.**

***Index Terms*—Distributed Computing, Matrix Multiplication, Multi-node Cluster, Mobile Phones, Performance Analysis, Offloading**

I. Introduction

If we compare the computation power and services offered by smart phones these days with smart phones released a decade ago, we observe several drastic changes. The services offered and the computation power promised by the smart phones has increased exponentially with the advancements in microchips and storage device technologies. Network sensors are more stable, reliable and provide higher bandwidth and speed. On the other side, the computation power offered by personal computers is also growing. Nonetheless, smart phones stand out as promising candidates for solving considerably involved computational problems.

A. Relevance to Mobile computing

Considering the computational power offered by an average smart-phone, it seems quite possible that the smart-phone can be used to run a part of a complex computational task. Expanding on this premise, we try to implement a distributed system consisting of only smart-phones to solve a mathematically complex problem of matrix multiplication. The distributed infrastructure works in master-slave architecture. Slaves also known as workers share their battery stats, location information and charging status. Based on these parameters, the master note decides the amount of work assigned to the worker node. If few worker nodes fail to do the computation for any reason like network disconnected, master node reassigns the failed computation tasks to available worker nodes, making the system highly resilient and fault tolerant.

II. System Architecture

We implemented a master-slave distributed architecture in such a way that communication happens to and fro between master node and slaves. No communication will happen between worker nodes. There is a single master node and there can be more than 1 worker nodes.

A. Setup

We have developed an android mobile application, which provides the user with an option to start the application in master mode or worker mode, selecting both choices is not allowed by design. If the user selects to make the device act as master node, the application tries to search for available workers. On the other side, if the user selects to make the device act as a worker node, the application goes into discovery mode and makes itself available to be found by the master device running the same application. We use the Nearby Connection API for achieving peer to peer connection between smartphones which are in the vicinity of each other. We use Wi-Fi connection if it is available, else we fall back to Bluetooth. Devices which are not close to each other are ignored from the computation by master node by design.

B. Permissions

Once the user opens, the application for the first time, the user is asked to grant following permissions:

1) Access to the device storage (Photos and Media folders)
2) Access to the device location (Fine Accuracy)

These permissions are required by the application at various stages of the application life cycle.

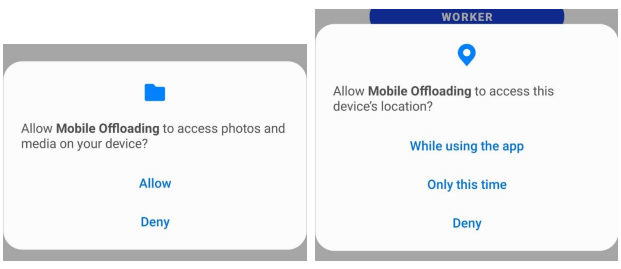

Fig. 1. Permission request to device storage and location

III. IMPLEMENTATION

The android application consists of two modes, Master mode and Worker mode. The following sections describe in detail about the application flow in both the modes.

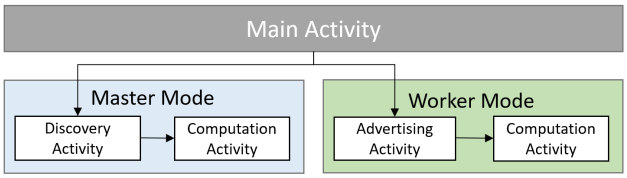

Fig. 2. Application Architecture

A. *Smartphone acts as a Worker*

Initially on starting the application the user is asked to choose between the two modes in the main activity. Once the user chooses to run the application as worker mode, the application navigates to the next activity in which the smart-phone application runs as an advertiser, where it advertises its presence to nearby devices running the same application context. The strategy adopted to advertise is the ”P2P_CLUSTER” strategy. In this strategy the many devices in the worker mode can connect to master within a range of 100 meters. Both of the devices can initiate the connection and start communication.[3]

Once the device is successfully advertising, we show the indication in the user interface that it is visible to all the masters nearby. We also show the statistics of the device battery statistics such as the battery percentage, if the device is plugged in to the power outlet or not.The device location information [2] (latitude and longitude) is also shown on the user interface. These statistics are refreshed at every few seconds.

In the next phase once the device is found by the master. The master device will initiate a connection to connect to worker. Here we show a android notification in to ”Accept” or ”Reject” the connection. If the connection is rejected, nothing happens in the worker and the device is still visible to master. If the user accepts the connection, then an acceptance is sent and a communication channel is established once the connection is successful. After the acceptance the application navigates to the next activity, ”Worker Computation”. Here the worker uses the previously established communication channel to send and receive the payloads.

In the worker computation phase, the worker waits for the master to send the payload data. Until then there is a busy indication shown on the screen with a message ”Waiting for work”. Once the worker receives the work, it computes it and then sends back the result via the communication channel.

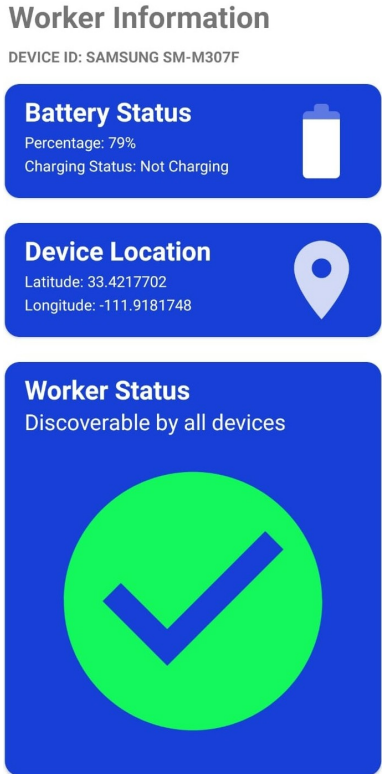

Fig. 3. Worker advertising screen

This communication continues until the master sends a work complete message.

B. *Smartphone acts as Master*

If the user starts the application as a Master, the application runs in a discovery mode. Nearby workers who are sharing their presence via message payload on Wi-Fi or Bluetooth are detected by the Master application. Master application maintains a list of such workers and sends them an connection request. If the worker rejects the computation task, the entry of the worker device is removed from the list which is maintained on the Master device. On the other hand, if the worker accepts the computation task by clicking on accept in the notification, the worker is added to the connected devices list. The worker sends current battery level, whether the device is connected to charging station and device location co-ordinates with the master at every few seconds. When a desirable number of worker devices are available and have accepted the computation task in-app notification, Master shows a button to ’Assign Task’ at the bottom of the Master app screen. After clicking on this button, the master device moves to a new activity called ”Master Computation”.

In this activity, master performs a matrix multiplication task locally and notes down the time taken by the task and displayed on the screen in the master computation activity. After the task is finished, the same matrix multiplication task is sent in a distributed fashion by the master across available worker devices. Workers also move to a computation activity

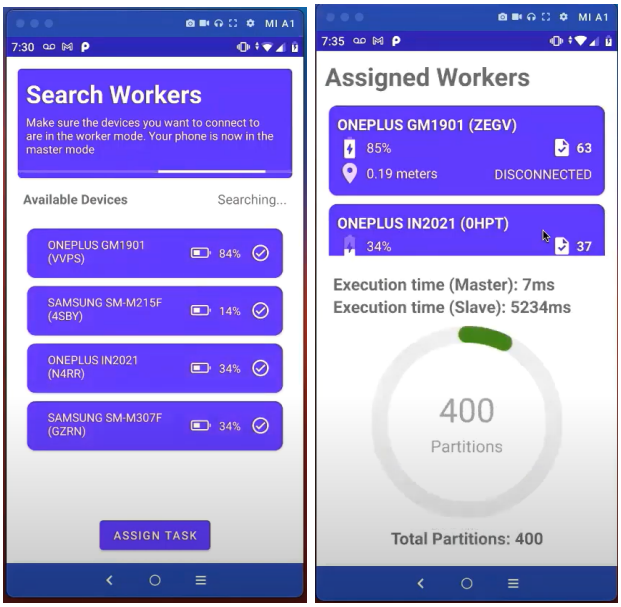

Fig. 4. Master Discovery(left) and Computation(right) screen

and show the screen stating that the distributed computation is in progress. Master maintains and manages these tasks and shows a progress bar informing about the current status of matrix multiplication task.

a) Distributed Computing: Once the master computes the matrix multiplication on the same device, The next step is to distribute the task among the available workers. The strategy for allocating the work is performed using a priority queue.

Matrix Multiplication Partition Process: For multiplying the two available matrices and obtain results, logically we need to find the values of each cell of matrix which is defined by matrix[row][column]. We call these cells partitions. Since we are using two square matrices for multiplication, we define a Hash Table with key as partition index and value as the value of the cell in resultant matrix. For example,

$$\begin{bmatrix} x_1 & x_2 \\ x_3 & x_4 \end{bmatrix} \times \begin{bmatrix} y_1 & y_2 \\ y_3 & y_4 \end{bmatrix} = \begin{bmatrix} z_1 & z_1 \\ z_3 & z_4 \end{bmatrix}$$

For above matrices, the partitions will be:

$$z_1 = x_1 \times y_1 + x_2 \times y_3$$
$$z_2 = x_1 \times y_2 + x_2 \times y_4$$
$$z_3 = x_3 \times y_1 + x_4 \times y_3$$
$$z_4 = x_3 \times y_2 + x_4 \times y_4$$

After we determine the partitions, we split these partitions and send the data i.e. rows and columns required for calculating a each individual resultant cell, across devices. We can achieve this distribution because of mathematical property of matrix multiplication that, Dot product of the individual input columns and row results in the cell of resultant matrix. Thus, we send following partitions across the devices. The result of z_1 is available in partition 1 and the result of z_2 is available in partition 2 and so on.

$$partition_1 = [[x_1, x_2], [y_1, y_3]]$$
$$partition_2 = [[x_1, x_2], [y_2, y_4]]$$
$$partition_3 = [[x_3, x_4], [y_1, y_3]]$$
$$partition_4 = [[x_3, x_4], [y_2, y_4]]$$

For achieving fault tolerance in our distributed system, We decided to go ahead with the system design of granular data sharing across workers using the above mentioned partitions. With this implementation, we avoid the master from sharing huge chunks of data with workers. Considering any worker can fail while computation or the worker can disconnect at any time, having a granular data sharing makes it easy to redistribute the failed tasks to other workers. This partition methodology, also avoids sending redundant data across partitions keeping the shared data across workers to minimum. We initially populate all workers in an Arraylist. From this workers Arraylist, we generate a Priority-Queue of workers with comparison condition as follows:

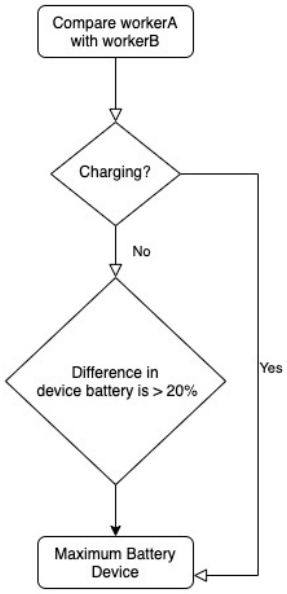

Fig. 5. Compare function for Priority Queue

Using the above implementation, we achieve selection of workers with more battery and workers which are currently docked to charging stations over devices with lesser battery. Here, we also check check if the difference in battery levels is greater than 20. If that is the case, we add device with more battery first in the priority queue. Also, we ignore worker devices with battery level less than 15Once the priority queue is computed, based on the priority we send workers computation tasks. For achieving this functionality, we use the Hashmap of partition indices. One computation consists of required row and column values required for calculating

the value at partition index in the resultant column. Once the data sent to these devices our implementation removes the worker from the priority queue. Once the computation result is obtained for the partition index from the worker node, we add the worker again in the priority queue. While adding worker in the priority queue, again the comparator function kicks in and adds the worker in correct desired position in the priority. Using this mechanism, we iterator over all the elements in the priority indices Hashmap and obtain values for each partition index. If the value for the partition index key is present conveys that the computation is completed for that partition index and we do not need to compute it again. If the value absent for the partition index in Hashmap, means the computation task is not complemented. So, if a worker disconnects and looses connection with the master, the computation task is lost. Since there is no entry for that partition index in the Hashmap, while iterating we send the computation task to other available workers. The advantages of this implementation are multi-fold. It allows any worker to go down in the distributed environment and still achieves task completion making the infrastructure highly fault tolerant.

Continuous Device Monitoring Strategy: We monitor the worker device statistics like battery level, is the device charging and longitude, latitude co-ordinates by sending a data payload from worker to master every 5 seconds. If the statistics payload or data message with computation task is not received on the master side for 10 seconds, we consider that the worker has encountered a fault and remove the worker from the list of available workers for the computation. Thus, effectively using these payload data messages we achieve heart-beating messages between all available workers and master node.

b) Failure Recovery: Once we send computation data payload to worker, it gets removed from the priority queue. Afterwords, when worker sends computation task result back to master, we add the worker again in the priority queue of available workers. In this step, if a worker gets disconnected from the network, we do not receive any message from the worker which includes device stats or data payload. If that occurs, we remove the worker from the list of available workers and that worker never appears in the priority queue which is used to send tasks to available workers. This computation task, since it does not have any value computed in the partition index Hash Map, gets calculated again using other available workers.

IV. Performance Analysis

We agreed upon measuring the performance of our information using below metric:

a) Power consumption: This metric takes into account how much battery is consumed by master and worker nodes for the actual computation and sharing data messages across the between the master node and workers. Our measurement also includes the power consumed by Wi-Fi, Bluetooth and location tracking.

Execution time	Time (ms)
Master only	7
Workers no failure	4170
Workers failure scenario	5234

TABLE I
EXECUTION TIME FOR A MATRIX MULTIPLICATION OF 20×20 MATRIX

b) Time taken for computation: This metric measures the time taken for the computation. This metric includes initialization time, time required for sending data over network and actual computation task. We measured the time taken for computation for two cases.

1) Time taken for only computation on the master side.
2) Time taken for computation on worker side added with aggregation of results on the master side. This metric also includes communication

overhead incurred between the master and worker devices.

V. Observations

We measured observations about our distributed system implementation of mobile phones using the performance analysis metric defined above.

a) Power consumption metrics: For a matrix multiplication of 20×20 matrices, the battery consumed on master node is below 1%. For Matrix multiplication done solely on the worker, no significant battery drop in the master battery was observed. For the distributed computation workers observed less than 1% battery consumption.[1]

b) Execution time metrics:

1) Matrix multiplication of 20×20 size was completed on master node within 7 milliseconds.(See table 1)
2) Matrix multiplication on distributed environment took around 5000 milliseconds.

when 2 devices were disconnected from the network when the computation was going on, we observed that time taken increased by 25% amount.

VI. Limitations

In our distributed computation model, When the computation task is very less it fails to obtain better results than local computation. Thus, in order to make mobile distributed network shine, we need to have a computation task that takes long time to finish on a single node. Once the master assigns the tasks to the workers and the computation has started, Even if there is a new worker is available, it cannot assign the task to it and the worker has to wait for the next time master tries to discover it.

Conclusion

In this project, we could successfully achieve implementation of a distributed system on a number of mobile phones which are connected to the same network or are in the Bluetooth vicinity of each other. We leveraged this distributed system infrastructure to implement matrix multiplication across number of devices. The design we implemented is a highly resilient and fault tolerant by design. Using various matrix sizes

Sl.No	Task Name	Assignee
1	Master Discovery activity (Searching/Listing of Devices)	Anvesh, Midhun
2	Worker Advertisement activity (Device advertises its availability)	Anvesh, Yuchen
3	Master Connection Establishment service (Master connects with Worker)	Anvesh, Midhun
4	Worker Connection Establishment service (Worker connects with Master)	Anvesh, Sai
5	Device Stats Broadcast service (battery/location details sent to Master)	Sai, Midhun
6	Device Stats Monitoring service (Master monitor the device stats of all workers)	Anvesh, Midhun
7	Master activity (work distributed to all workers and results are combined)	Anvesh, Sai, Midhun
8	Failure recovery algorithm (redistribute work to another worker incase of failure)	Midhun, Sai
9	Master execution time estimation algorithm (compute master alone execution time)	Midhun, Yuchen
10	Distributed execution time estimation algorithm	Sai, Anvesh
11	Master power consumption algorithm	Midhun, Anvesh
12	Worker power consumption algorithm	Midhun, Anvesh

TABLE II
TASK DISTRIBUTION TABLE

for computation we observed that in a distributed environment with mobile phones, if the computation task is not heavy, then the overhead of communication outweighs the benefits of distributed computing making time to reach results longer. On the other hand, for a considerably large matrix size, distributed computation is faster than the computation which is done on a master device.

REFERENCES

[1] Android API. *Battery Manager*. [Online; accessed 25-March-2021]. 2021. URL: https://developer.android.com/reference/android/os/BatteryManager.

[2] Android API. *Location Data*. [Online; accessed 25-March-2021]. 2021. URL: https://developer.android.com/training/location/request-updates.

[3] Google API. *Near By Connections*. [Online; accessed 25-March-2021]. 2021. URL: https://developers.google.com/nearby/connections/overview.