Parallel Computing on Android Platform

Divesh Soni
Department of Computer Science
Rochester Institute of Technology

Abstract-Mobile phones are being incorporated with increased processing capabilities to handle more and more processor-intensive applications. However, battery and storage limitations inherent with mobile computing still prevail and require mobile application developers to utilize available resources on the mobile platform with caution. Computational off-loading framework can be a viable option to ease off the usage of available resources. It is a common practice to employ cloud services to handle the off-loaded tasks to augment the limited resources on mobile devices. In this paper, leveraging the resources of other available mobile devices in the vicinity is proposed as an alternative to using cloud services to process the distributed workload. An android application for sharing images is developed and the performance of cloud services is compared with roping in other devices in the vicinity to accomplish a relatively computationally-heavy task of face recognition. Analyzing the performance of this Android application, it is observed that using computational power of other mobile devices around is efficient in terms of computational time and battery consumption, as compared to using cloud services.

1. Introduction
Parallelization is becoming an integral part of architecture design, in case of systems ranging from mobile phones to cloud-based enterprise solutions. The processing power possessed by the computing systems being used these days is increasing by and large [6], with even mobile phones incorporating four to eight processors becoming a norm. However, even after these progressions in terms processing abilities of the mobile systems, there exist intrinsic constraints which limit their processing capability, power and memory to be scaled beyond a certain threshold.

The areas of distributed systems and parallel computing have been topics of extensive research [1]. In recent times, these areas have been explored in context of mobile computing, given the advancements in multicore systems and cloud computing platforms that can assist mobile systems. However, relatively not as much research has been focused on trying to make use of the available mobile resources in the environs. This can be critical considering the extent to which mobile phones have become ubiquitous.

In this paper, the work off-loading framework to utilize the surrounding mobile devices' resources in parallel is evaluated. The performance when compared to off-loading the same task to the cloud is compared to clearly determine the performance gains of this parallel computing approach. Addressing some of the critical issues like reduced battery consumption and lower computational time that are tied to mobile computing, is the principal motivation driving this experimental analysis.

An Android application is developed which would perform face recognition in parallel on different devices in the vicinity. The goal is to fetch images of a certain person on other devices that are present around. The
devices communicate directly with each other to transfer the images after the face recognition. Alternatively, to analyze the performance of face recognition process being off-loaded to cloud, a web server is developed in Java which will receive all the candidate images and the target image to subsequently perform face recognition. The matched images after the face recognition process will be sent from the server to the initiating device.

The results from evaluating the performance of the parallel computing and the cloud computing approach to tackle the face recognition problem as mentioned above, show that the parallel computing approach has a significant performance advantage. The parallel computing approach is better in terms of time taken to complete facial recognition of a fixed set of images and also involves less battery consumption. The size of the set of images considered for facial recognition was varied and results were analyzed for each set size.

The rest of the paper follows the structure like so. The next section outlines the literature review to shed light on related research papers and articles. Section 3 discusses methods and techniques used in this paper followed by Section 4 which gives the implementation details. Test and results are provided in Section 5. Section 6 discusses the future scope of this paper and finally Section 7 summarizes the results and provides a conclusion.

2. Literature Review

To cope with the limited resources available on the mobile platform, it is imperative for application developers for the mobile platform to follow architectural styles and best practices to preserve the durability of the resources. [6] suggests coping with this limitation of mobile computation with delegation of the tasks to be performed to the cloud infrastructure which has considerably more resources to handle such delegated tasks. The approach to be followed for off-loading can be effective if the correct modules of the applications are chosen to be off-loaded and retained. Determining the most computationally-intensive modules is a key here. Performing this process in an asynchronous manner is necessary in order for the users of mobile application to carry on with other available features that are provided by the application. Such off-loading of tasks should be done with caution since it does involve a trade-off between using up the battery and communication bandwidth as opposed to local processing resources. The decision to off-load to cloud should be well-informed and the associated risks should be determined beforehand.

[2] advocates that the notion of crowdsourcing can be extended for use on mobile platforms in order to utilize the processing capabilities of a set of devices available in the vicinity. This approach is based upon the idea of load balancing by using a variation of the work stealing method. This opportunistic off-loading happens in real-time with no pre-determined centralized unit acting as a coordinator when the need for off-loading arises. Such opportunistic off-loading might turn out to be complicated to handle because of the need to transfer the code and state information.

The Android application developed for image sharing and face recognition for the experimental analysis builds upon the idea suggested in [2] and distributes the task of face recognition among the available surrounding devices. Moreover, it does not involve any state information to be consistent with all the devices involved, it merely requires the initiating device to send first target image to
kick off the face recognition process on other devices.

3. Methods and Techniques

Image sharing would be kicked off by a user sending the image of the target person to available devices, which will be followed by those devices performing face recognition algorithm on the images available locally. The images that will be having the target person in it would all then be sent back from all the devices to the initiating device. This follows a master slave architecture wherein the initiating device acts as the master and the other involved devices act as slaves.

Kairos Face Recognition and Human Emotion Analytics API is used \[3\] for the process of facial recognition. It is an open-source API available for free use with certain limits on the number of query requests being sent. Being an open source service, Kairos provides security by providing a security key and app id to authenticate each request sent to the API. The two primary features of the API that were used by the developed Android application are registration and recognition.

Face registration involves inserting one or more images of the target person in the face database. Each person will have a subject ID associated with them, which is used for registering their image in the face database. More the number of images associated with a specific person, better is the accuracy to recognize that person later on when recognition requests arrive. The API recommends having at about 8-10 images for each subject. All of the images are mapped to a unique subject ID and all the related subjects are grouped together within galleries, each of which is identified with a unique gallery ID. The structure of the face database is illustrated as Fig. 1 below.

![Fig. 1: Structure of face database](image)

Face Recognition involves querying the face database with an image to check if that image matches any subject already registered in the face database. Each recognition request would provide a gallery ID along with the image which starts the recognition process with all subjects within the specified gallery.

4. Implementation

I. Architecture

The architecture for the Android application developed for image sharing is as outlined in Fig.2 below.

![Fig. 2. Architecture for image-sharing application](image)
by all the slaves connected that are in the surroundings. Following this, master waits to receive images from the slaves that contain the target subject. On the other hand, each slave would start with registering the received image from the server in the face database. This would be followed by performing face recognition and sending the matched images to the master in parallel. The face recognition is performed on the N most recent images available on slave’s local memory. The value of N is varied to experiment with different image subset sizes and monitor the performance of the framework.

II. Image compression

The images captured by today’s high-end cameras incorporated on mobile phones are very high resolution and hence take up lot of memory space. While this provides the user with a good quality image, it is memory-intensive and time-consuming to transfer them to other devices. Thus, before transferring images, they are converted to a standard PNG format. This ensures uniformity and also allows retaining a good quality image on the receiving end since PNG facilitates lossless compression and decompression. Moreover, the images stored in the form of Bitmaps are converted to byte arrays for quick transfers across the devices.

III. Scanning for slaves

In order to scan for devices in the vicinity which would be available to perform face recognition and share the respective images, the master creates a multicast group. The process of scanning for slaves can be illustrated as shown in Fig. 3.

The slaves can join the group and allow retrieval of master’s IP address which can then be used to setup a TCP connection from each slave. The subsequent transfer of images is done over this established TCP connection instead of UDP due it’s reliability over UDP.

5. Evaluation

The performance of the parallel computing and cloud computing approaches to handle the off-loaded task of face recognition is evaluated. The results of these evaluations are then compared to determine the performance gains and tradeoffs associated with either approach.

I. Evaluation Settings

The performance of the Android application is tested using varied size of image sets, which correspond to the potential images that need to be tested for face recognition. Each slave device contains this set of images within it locally. Android phones used for testing purposes include 3 LG Nexus 5 and one Moto E, where one of the Nexus 5 acts as master and the rest act as slaves. Also, master and all the slaves are required to be on the same network in order for them to establish the connection and communicate with each other. In case of evaluating the cloud computing approach, the webserver and the
Android devices involved need to be on the same network.

II. Results

Fig. 4 shows the results of comparing the time taken by both – parallel computing and the cloud computing approaches in terms of time taken to perform face recognition of varied number of images.

As observed from the above line graph, the time taken to perform face recognition by both the approaches is not too different for a subset having as little as 10 images. However, as the number of images that undergo face recognition increase, the cloud computing approach takes far more time to complete the facial recognition of the same number of images than the parallel computing approach. The experiment is performed for 10, 50, 100, 150 and 250 most recent images available on each slave device.

Fig. 5 shows battery consumption vs number of images for both parallel computing and cloud computing approach.

As observed from the line graph in Fig. 5, the battery consumption is not very different for both the approaches, regardless of the number of images that undergo face recognition. Especially for small number of images when undergo face recognition, there is absolutely no difference at all in terms of battery usage as observed the results for 50 and 100 images. As the number of images increase, parallel computing tends to do a little better than its cloud counterpart.

Additionally, there were experiments performed with increasing the number of devices involved in the process, but it showed negligible difference in terms of both time and battery consumption for both the approaches. The number of devices used for experimental analysis was increased from 4 to 7 to evaluate their performance.

6. Future Scope

The Android application programmatically sets the number of locally available images that will be considered for face recognition. For instance, if that number is set to 50, then application would select 50 most recent images on the device and queue them for face recognition. It would be helpful if the user can provide the number of images from the interface instead of it being set programmatically. This way user can freely
control the usage of their battery and processing power by setting them manually through the user interface.

The developed Android application involves considerable amount of data transfer but it has no security mechanisms implemented. Some form of authentication layer could increase the participant slave device’s confidence in using the application. Since transfer of images is such a crucial part of the application, ensuring the integrity of received images is imperative.

7. Conclusion

In this paper, off-loading to other mobile device is presented as an alternative to off-loading to cloud platform and their performances are compared. It can be noted that since this image sharing application involved many more images being transferred to the cloud whereas using the resources of other devices locally would considerable reduce the number of images being transferred. Considering the results from the experimental analysis, this increase in image transferred incurred additional overhead causes the cloud computing approach to consume more battery and computational time. Thus, depending on the application under consideration, off-loading work load to other devices could give significant performance boosts as opposed to using the conventional cloud services.

References