Towards Energy Efficient Real Time Machine Learning on Smartphones

Habib Sabiu
University Of Saskatchewan
Saskatoon, Canada
habib.sabiu@usask.ca

ABSTRACT

Limited battery capacity and computation power of mobile devices hinder the advancement of many user contextual applications that requires continuous collection and processing of sensor data in real-time using machine learning algorithms such as user activity recognition. This is because continuously capturing and processing of sensor data on mobile devices consumes a huge amount of energy. To investigate the energy overhead of such continuous activity sensing, this paper analyzes the performance of real-time execution of machine learning algorithms on mobile phones for activity recognition. We proposed the use of computation offloading from mobile phones to remote servers as a viable means to save both mobile phone battery power and computation time. Through experiments, we analyzed the impact of computation offloading in terms of energy savings and computation time. We develop an Android based application for collecting sensor data, and a remote server that uses K-Nearest Neighbor algorithm to detect user activity in real-time. To evaluate the amount of energy and computation time save using computation offloading, we compare performing continuous activity recognition on the mobile phone with offloading the computation to a remote server in terms of CPU load, battery power usage, and computation latency. Experimental results show that a large amount of battery power consumption and computation time can be minimized by having mobile devices offload computation to a remote server.

1. INTRODUCTION

The features and complexity of mobile phones have increased over the years. The ubiquity of mobile phones, their size, computation and sensing power, ability to send and receive data, low cost, and many useful applications makes it easy for people to acquire and use them for their day to day activities. Today mobile phones are equipped with complex sensing capabilities such as Accelerometer, Gyroscope, Magnetic Field, GPS, WiFi, Bluetooth and so on. These embedded sensors have become a rich data source to measure various aspects of a user's daily life, and are the drivers of many context-aware applications that takes advantage of user contextual information to deliver rich and personalized services. Such services could be location aware, such as real-time traffic monitoring, navigation systems, indoor localization, and location-based social networking. Others are health related applications that aim to track healthy lifestyle in individual e.g. activity recognition, step counting used in pedometer applications, health and fitness monitoring and so on. A more recent application is the use of machine learning, which are set of methods used to explore a large data in intelligent ways to find patterns, and used the uncovered patterns to predict feature data. Thus, mobile phones are no longer only communication devices, but also a powerful tool for environmental sensing that can monitor a user's ambient context both efficiently and in real time without much effort from the user.

However, recognizing user context from sensor data requires sensing and storing of large data collected at a high frequency often in milliseconds. Processing this data to make inference using machine learning algorithms requires large computing and storage capacity which is often not available on mobile phones. Another major drawback to context detection is the limited battery capacity of mobile devices. The embedded sensors in mobile devices are the major consumers of battery power. Turning these sensors on all the time to sense user context drain the phone battery quickly. Over the past years, many energy savings techniques for mobile devices have been proposed in the literature. Smart batteries, power scheduling, efficient operating systems and applications, efficient graphical user interfaces, energy-aware communication protocols, and task offloading are all examples of these methodologies and techniques [13]. In most recent studies, energy was saved by collecting sensor data within a time frame. For example, the sensors are turned on for some minutes to collect data and then turned off. Data collected is stored on the phone for a specified time before being offloaded to a server when the phone is plugged into a power outlet to charge and connected to a WiFi. On the server side, the data is processed using some data processing techniques such as machine learning or stored for later analysis. While these approaches save battery, it hinders the advancement of many user contextual applications that requires collection and processing of sensor data in real time and displaying the results on the mobile device. An example of such application is the use of machine learning on sensor data to detect user activity in real time e.g. standing, sitting, walking,

running, walking upstairs, walking downstairs. Such user activity detection has many health related applications such as tracking healthy lifestyles in individuals.

To investigate the energy overhead of such continuous activity sensing, this paper analyzes the performance of real-time execution of machine learning algorithms on mobile phones for activity recognition. An activity recognition application takes a raw sensor reading as inputs and predicts a user's motion activity using statistical analysis or machine learning. The project uses K-Nearest Neighbor algorithm (KNN) on Accelerometer, Gyroscope and Magnetic Field sensor data to detect user activity including standing, sitting, walking, running, walking upstairs, walking downstairs in real time. KNN is a supervised learning instance-based classifier algorithm based on the majority voting of surrounding neighbors. Using experimentation, the project shows sensor data can be collected and processed using machine learning in real time. However, a large amount of battery power and computation time can be saved by having mobile devices offload computation to cloud servers for processing. In this type of settings, data is collected on the mobile device in real time and offloaded to a remote server where the machine learning algorithm is executed and the results of the computation are send back to the mobile device for display, a process known as computation offloading. With the advancements in networking technologies, computation offloading has proving to be an effective means of saving energy on mobile devices. The project compares energy consumption cost and latency of performing local data analysis on mobile phones versus performing data analysis on the back-end server but pays for the data communication energy. Specifically, compute versus communication cost were analyzed in terms of CPU load, battery consumption, and processing time on mobile devices.

2. BACKGROUND

This section gives a brief background on the different concepts used in this project. The section begins with an overview of Android mobile sensors. Then give an introduction to computation uploading, and the machine learning algorithm (KNN) used in this project for activity detection.

2.1 Mobile sensors

Sensors are the source for raw data collection in activity recognition application. Most Android devices have built-in sensors that measure motion, orientation, and various environmental conditions. These sensors are capable of providing raw data with high precision and accuracy. The sensors are also useful in tracking three-dimensional device position or changes in the device ambient environment [2]. There are three categories of sensors supported by the Android platform. These includes: *Motion sensors* that measure acceleration and rotational forces along three axes (x, y and z) as shown in Figure 1, e.g accelerometers, gravity sensors, gyroscopes, and rotational vector sensors. *Environmental sensors* that measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity, e.g. barometers, photometers, and thermometers. *Position sensors* that measure the physical position of a device, e.g. orientation sensors and magnetometers. This project combines data collected from two motion sensors (Accelerometer and Gyroscope) and a position sensor (Magnetometer) to detect user activity in real time. Previous research [15] [16] shows combining multiple sensors increase the accuracy of the activity detection.

Accelerometer is a hardware sensor that measures the acceleration force in m/s^2 that is applied to a device on all three physical axes (x, y, z as shown in Figure 1), including the force of gravity. This sensor is commonly used in motion detection.

Gyroscope is a hardware sensor that measures a device's rate of rotation in rad/s around each of the three physical axes (x, y, z as shown in Figure 1). This sensor is commonly used in rotation detection.

Magnetometer is a hardware sensor that measures the ambient geomagnetic field for all three physical axes (x, y, z as shown in Figure 1) in μT.

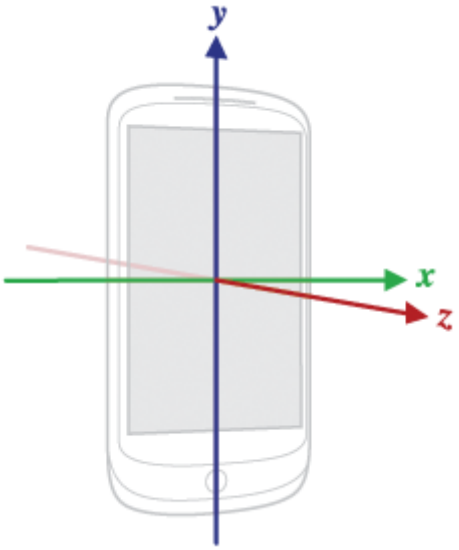

Figure 1: Coordinate system (relative to a device) [2]

2.2 Computation uploading

Mobile computational offloading originated from the area of distributed systems where components located on multiple networked computers communicate and coordinate their actions by passing messages. Many protocols have been successfully deployed and used over the years to call a method or procedure that resides on a remote node. A remote method is invoked by calling the remote method signature together with the necessary input data to the method. Example of such technologies used in the area of distributed computing are: Remote Procedure Call (RPC), The Common Object Request Broker Architecture (CORBA), Simple Object Access Protocol(SOAP), Representational State Transfer (REST), Remote Method Invocation (RMI), Remoting, Message Passing Interface (MPI) and so on. However, those protocols are usually designed for wall powered computing devices connected over networks. Thus they focus on increasing performance, fault tolerance, and accessing remote resources rather than energy issues. On the other hand, mobile devices are energy limited with limited network bandwidth and highly inconsistent connectivity. Due to these

reasons, many researchers [3][12][5][7] exploit the possibility of computation offloading approaches targeting mobile devices.

2.3 K-Nearest Neighbor

Previous studies [9] used K-Nearest Neighbor algorithm on mobile sensors data to detect user activity. K-Nearest Neighbors algorithm (KNN) is a simple non-parametric machine learning algorithm used for classification. The input to KNN algorithm consists of the k closest training examples in the feature space and the output is a class membership. KNN stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions). An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors measured by a distance function. If k = 1, then the object is simply assigned to the class of that single nearest neighbor. KNN is a type of instance-based supervised learning, where the function is only approximated locally and all computation is deferred until classification.

3. RELATED WORK

Computation offloading from smartphones to the cloud has been an active area of research over the past years. Many researchers proposed using computation offloading as a viable means to save energy on mobile phones. Although computation offloading save energy, it introduces communication cost. Therefore, the consideration of the communication cost is crucial for the effectiveness of computation offloading. Altamimi et al. [3] developed energy estimation models to compare the cost of performing computation locally (on the smartphone) with offloading the computation to a remote server. Their research focuses on developing energy models to estimate the cost of task offloading caused by the networking activities. In their model, computation is offloaded to the cloud if less energy is consumed by communicating between the mobile device and the remote server compared to local processing. A similar research by Lee [12] present a framework for efficient and seamless offloading of computation to a remote server. Their framework uses energy efficient static analysis and object serialization approach to profiles an Android application's internal code including Java class methods and user-defined native methods for a short period of time. Using the profiling data, their framework automatically identifies target methods for remote execution. The static analysis identifies a minimum set of data elements that are necessary for remote execution thereby shrinking the size of data transferred to the server. The server also optimizes the amount of data it sends back to the mobile phone by eliminating data transfers of unmodified data. Chen et al. [5] provides a programming framework that utilized aspect-oriented programming to upload part of computation from mobile phone to a cloud server. Their framework leverages the power of aspect-oriented programming to inserts appropriate offloading code into the source code of the target application based on the result of static and dynamic profiling. Therefore automating the offloading process. Cuervo et al. [7] present a system that enables fine-grained energy-aware offload of mobile code to the cloud. Their system uses properties of Common Language Runtime to automatically identify potential methods that can be remotely executed to save energy. And extract only the program state needed by those methods. Using method profiling and serialization, the system determines the communication cost for offloading these methods to a remote server. Thereby maximizing the potential for energy savings through dynamic fine-grained code offload while minimizing the changes required to the applications.

Due to the increase in availability of sensors on smartphones, activity recognition has gain much attention over the past years. Previous research [4][11] studied the use of accelerometer sensor found in modern smartphones for activity recognition. Basically, accelerometer sensor data are collected on smartphones while users go about their daily activities. A label for the current physical activity performed by the user is assigned to each of the collect records. These data is used as training samples for predictive models. The data is transferred to a server either directly by connecting the mobile phone to the server using USB or over the network (cellular and WiFi). On the server side, the data is processed using conventional classification algorithms such as naive bayes, decision trees, logistic regression and multilayer neural networks. Other works [15] [16] combine multiple sensors such as accelerometer, gyroscope, and magnetometer to improve the accuracy of the machine learning classification algorithm thereby making the recognition process more reliable. While these works explore the possibilities of recognizing user activity based on sensor data, the activity recognition is not real time. Rather, a large sensory data is collected and transferred to a server where classification algorithms are applied on the collected data offline, using a large part of the collected data for training. Giving offline processing an advantage for achieving higher classification accuracy since there will be higher overlap between the training and the testing data. This clearly limits the advancements of many applications that require collection, analysis and classification of a new instance of sensory data in real time. In a similar study, Kose et al. [10] compare the accuracy of different classifier algorithm for online activity recognition based on accelerometer sensors embedded on smartphones. Their system collects accelerometer sensor data and performs both the training to learn model parameters in real time instead of processing the data offline, and classification of new instances on the phone. Their work focuses on the performance of different classifier algorithms in terms of accuracy, rather than performance evaluation of computation offloading in regards to energy efficiency.

A different research by Yan et al. [18] investigates the effect of accelerometer sampling frequency and classification features in terms of energy overhead and classification accuracy. They proposed an approach that is based on real-time (as individual goes about their daily activities) adaptation of both accelerometer sampling frequency and the classification features to minimize the energy overhead of continuous activity recognition using mobile sensors. Wang et al. [17] present a framework for an energy efficient mobile sensing that uses hierarchical sensor management strategy to recognize user states as well as to detect state transitions. By powering only a minimum set of sensors and using appropriate sensor duty cycles their system significantly improves device battery life. Other similar research on energy efficient mobile sensing proposes the use of heterogeneous multiple processors on smartphones and using low-power processors for continuous sensing of user context [14]. Profiling and

optimizing the classifier algorithms to fit into a mobile environment by overcoming the challenges of mobile systems such as energy, latency, and to achieve an optimal energy-latency-accuracy trade-off [6].

4. EXPERIMENTAL SETUP

The experimental setup for this project is divided into two: a client-side Android application was developed for the collection of sensor data, and a back-end server was written in Java for processing the collected sensor data using machine learning. The client and the server communicate using sockets. Weka 3.9.0 was used to implement K-Nearest Neighbors classifier algorithm. Weka [8] is an API written in Java that provides a comprehensive collection of machine learning algorithms and data preprocessing tools. The default configurations of Weka's implementation of KNN algorithm (called IBk in Weka) was used. The number of the nearest neighbors (k) was set to be equal to 1 and Euclidean distance was used as a distance measurement metric for the algorithm. A KNN classifier model was developed and trained offline in Weka. Publicly available sensor dataset was used for training and testing the correctness of the model using 10 fold cross-validation. This dataset contains Accelerometer, Gyroscope and Magnetometer data collected by Shoaib et al. [15] on Samsung Galaxy S2, together with the physical activity class label including walking, running, sitting, standing, walking upstairs and downstairs. The developed KNN model was imported and used on both the client and server for online activity recognition. It should be noted that the underlining classification model is not re-trained and validated online as more data and information becomes available to improve the accuracy of the classification. This could be a limitation since to re-train the model, a new model must be implemented, train and validated in Weka and then imported into both client and the server. Never the less, the main focus of this project is not on the accuracy of the classification model itself but rather on the possibility of performing energy efficient online machine learning on smartphones. Therefore, the focus is on the processing that takes place in order to classify new instance based on the trained model. Trepn Power Profiler [1] was used for the measurement of energy consumption and processor utilization on the client side. Trepn Power Profiler is a power and performance profiling application for mobile devices. It can be used to measure and displays battery power, CPU and GPU frequency and utilization, network usage (cellular and Wi-Fi), and can profile a single application.

The client-side Android application record Accelerometer, Gyroscope, and Magnetic Field sensor data at a low frequency of one record every second. Although sensor data collected at a higher rate provides more training data, it may also introduce more noise. Therefore, a higher sampling rate does not always lead to a higher classification accuracy. The Accelerometer data includes the acceleration along the x-axis, y-axis, and z-axis. These axes capture the horizontal/sideways movement of the user (x-axis), upward/downward movement (y-axis), and forward/backward movement (z-axis). Gyroscope sensor reports the rate of rotation of the device around the 3 sensor axes(x, y and z). Rotation is positive in the counter-clockwise direction (right-hand rule). That is, an observer looking from some positive location on the x, y or z-axis at a device positioned on the origin would report positive rotation if the device appeared to be rotating counter-clockwise. Magnetic field sensor (also known as magnetometer) reports the ambient magnetic field, as measured along the 3 sensor axes (x, y and z).

(a) Performing computation on server

(b) Performing computation on client

Figure 2: Client side GUI with profiling graphs

Figure 2 show the client side graphical user interface together with Trepn Power Profiler graphs. The application collects and display sensor data each second. The four buttons on the application control where the data processing should be performed i.e whether to perform the machine learning on the client or send the data to a server for processing. On clicking the "ONSERVER START" button, the application first turns on WiFi connection and connect to the remote back-end server using socket. It should be noted that both the client and the server are on the same network. After connecting to the server, the application sends the collected sensor data to the server each second. The server performs the computation and assign a class label to the received sensor data and send the class label back to the client for displaying. "ONCLIENT START" button first turn off the WiFi connection then perform the machine learning classification on the mobile device. There is a label that shows the time taken between sending the data and getting the class label back. This data is also recorded for latency comparison between computation on the client and on the server.

4.1 Data collection

The Android client application was installed and run on a Samsung Galaxy S4 smartphone equipped with 1.6 GHz Quad-core Cortex-A15 CPU, 2 GB of RAM, Li-Ion 2600 mAh battery and Android OS v5.0.1 (Lollipop). This phone comes with built-in Accelerometer, Gyroscope, Magnetometer sensors. The server program was run on a system equipped with 8 core Intel Core i7 processor, 8 GB RAM, and Ubuntu 16.04 LTS operating system. The client and the server programs communicate over WiFi using sockets. It should be noted that both the server and client are connected to the

same network. The phone was profiled and a baseline measurement was collected using Trepn Power Profiler. These measurements were collected when the phone is not running any application other than the OS and the profiling application and all network communication are turned off. The measurements contain CPU utilization as shown in Figure 3 and battery power usage as shown in Figure 4. As seen from the figures, the average CPU load on the phone was around 5% while average battery power usage was around $73mW$. These baseline measurements are used to compare the resource usage and energy consumption of the Android application while collecting data and performing computation either on the client itself or uploading the computation to a server and displaying the results.

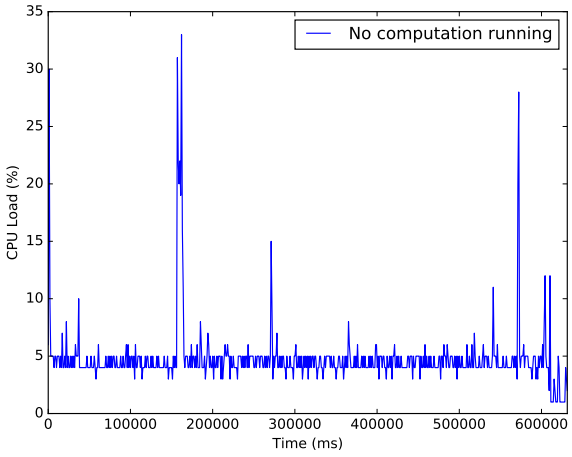

Figure 3: CPU Load Without Running Computation

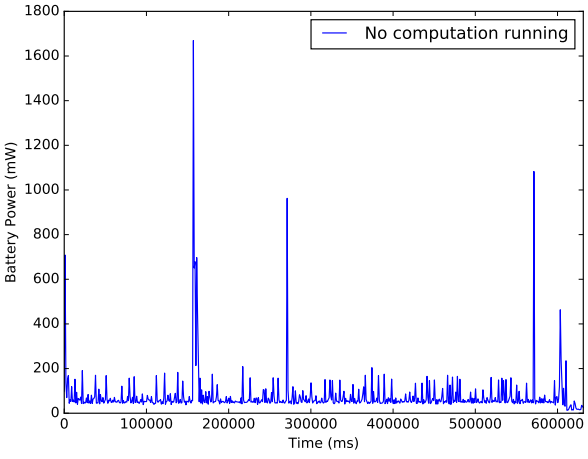

Figure 4: Battery Power Without Running Computation

A user was asked to carry the phone on their hand and perform different activities including standing, sitting, walking, running, walking upstairs, walking downstairs for specific periods of time while monitoring the activity recognized by the client Android application after classifying the collected sensor data using K-Nearest Neighbor algorithm (KNN). These activities were selected because they are performed regularly by many people in their daily routines, and involve the same continuous repetitive motions that often occur for long time periods, thus making them easier to recognize. Also, the training data used for this project contains labeled data for the selected activities. The data collection and activity recognition lasted for 5 minutes each for both on the server and on client processing. The process of data collection and activity recognition is shown in Figure 5. New record (accelerometer, gyroscope, and magnetometer) was collected and classified every second. The data collection interval was set to 1 second to reduce the amount of data collection and consequently minimize noise in the data. However, data collected each second should be sufficient to accurately classify the current activity a user is performing. During this period of time, Trepn Power Profiler application was used to profile the Android client application while performing the computation. This data is stored in a file on the phone for later analyses and comparison on resource usage and energy consumption of both performing computation on the client and on the server. The Android application also measures and record the time it takes between sending a computation request with sensor data and getting a response back. For example, when performing computation on the server, the application measure and record network latency for computation uploading. While for on client computation, the application measure CPU latency.

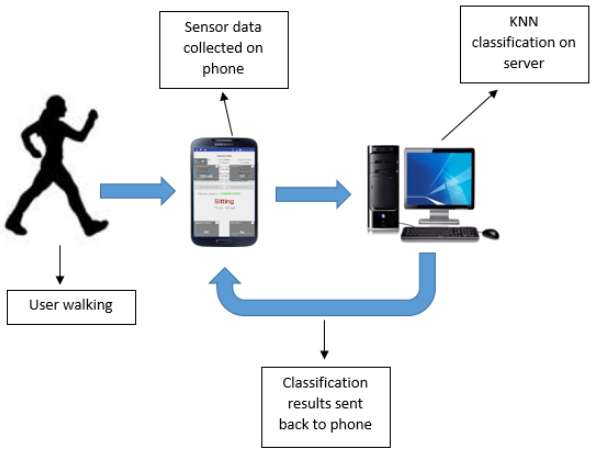

Figure 5: Data collection and activity recognition process using computation offloading

5. RESULTS

In this section, we analyzed the performance of computation offloading from mobile device to a remote server in terms of energy savings and computation latency. As discussed in the previous section, the Android application was first run for 5 minutes while collecting sensors data and performing activity recognition on the phone. All network connections (cellular and WiFi) were turned off since they are not being used to avoid additional load on the phone. CPU load, battery power usage, and the time it takes to perform KNN algorithm on the collected sensor data and display the activity recognized by the algorithm were recorded to a file every second. After the first 5 minutes, WiFi connection

was turned on and the activity recognition was performed on the server, the server sends results back to the client application for display. The same set of measurements (CPU load, battery power usage, latency) were recorded every second for 5 minutes while the computation is being performed on the server.

5.1 Impact on CPU load

We evaluate the impact of computation offloading on CPU load by comparing average CPU load for both on the client and on server computation with our baseline measurement. As shown in Figure 6, the average CPU load when performing activity recognition on the server is around 6%. An increase of about 1% when compared with the data collected when the phone was not running any application other than the OS and Trepn Power Profiler application. This shows that collecting sensor data and transferring it over the network have minimum impact on the average CPU load. On the other hand, when the activity recognition is performed on the phone, the average load raises to about 21%, making 16% increase in CPU load. This is expected since the classification algorithm (KNN) is now being executed on the phone, making a drastic increase in CPU load.

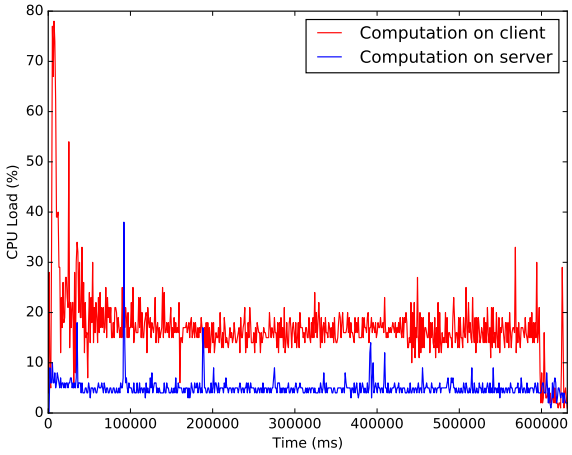

Figure 6: CPU Load With Computation

5.2 Impact on battery power usage

Figure 7 shows battery usage comparison between on client and on server activity recognition. An average of about $89mW$ battery power was used every second when executing the classification algorithm on the server. This account for only $16mW$ increase when compared with the baseline measurements. This shows collecting data from few sensors at a low frequency for example one record every second can minimize the amount of battery power usage. Also, WiFi network communication uses minimum amount of battery power. On the other hand, on client computation increase the average battery power usage to about $538mW$ per second. Accounting for $465mW$ increase compare to the baseline measurement, and $449mW$ increase in battery power usage when compared with performing the computation on the server. As expected, an increase in the CPU load leads to a drastic increase in battery power usage since CPU consumed a significant amount battery power on mobile devices.

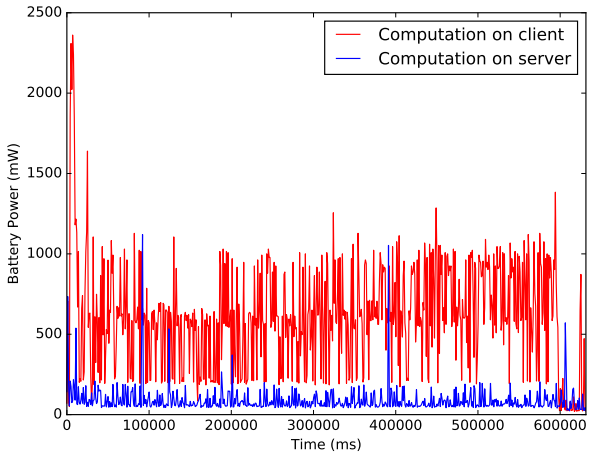

Figure 7: Battery Power With Computation

5.3 Impact on computation latency

Computation latency comparison between on client and on server activity recognition using K-Nearest Neighbor algorithm (KNN) on Accelerometer, Gyroscope, and Magnetic Field sensor data is shown in Figure 8. It takes $40ms$ on average for the Android client to receive classification response from the server after sending sensor data to the server. While it takes about $789ms$ on average to execute KNN algorithm on the client and display the classification results. This shows $749ms$ increase in response time when executing the activity recognition on the client.

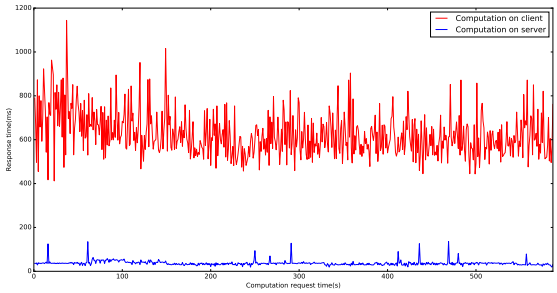

Figure 8: Computation Response Time

The results above shows that significant amount of battery power and computation time could be saved by taking advantage of computation offloading from mobile phone to remote server for many classes of applications that require collection and processing of mobile phone sensor data in near real time.

6. SUMMARY AND FUTURE WORK

Smartphones are now equipped with new and more sophisticated sensors that are capable of sensing user contextual information and their ambient environment in real time. This introduces new opportunities for many context-aware applications that deliver rich and personalized services. However, the energy consumption by these sensors and high computation requirement to process the sensor data in real-time

in order to make it useful, coupled with limited battery capacities of mobile devices, makes it almost infeasible to continuously sense user context and surrounding environment in real time. Therefore limiting the advancements of many useful application that require sensing user context in real time. In this paper, we analyzed the impact of computation offloading from mobile phone to remote server in terms of energy savings and computation time. We design an experiment by developing an Android based application for collecting sensor data, and a remote server that uses KNN algorithm to detect user activity in real time. We use three metrics: CPU load, battery power usage, and computation latency to evaluate the amount of energy and computation time save using computation offloading. To investigate energy and computation overhead, we compare on phone and on server continuous activity recognition in real time. Experimental results show that a large amount of battery power usage and computation time can be minimized by having mobile devices offload computation to a remote server.

In the future, we would like to extend the Android application to be able to re-train and validate the activity recognition model online and in real time as more data and information becomes available to improve the accuracy of the classification. We would also perform another experiment with multiple devices of different models connected to the server using cellular network such as 3G to investigate: 1) The effects of having multiple devices offloading computation to the same server. 2) The energy consumption and communication overhead of performing continuous activity recognition using machine learning on different smartphone models connected to a cloud server using cellular network instead of WiFi.

References

[1] Trepn power profiler. `https://developer.qualcomm.com/software/trepn-power-profiler`.

[2] Sensors overview. `https://developer.android.com/guide/topics/sensors/sensors_overview.html`.

[3] M. Altamimi, A. Abdrabou, K. Naik, and A. Nayak. Energy cost models of smartphones for task offloading to the cloud. *IEEE Transactions on Emerging Topics in Computing*, 3(3):384–398, 2015.

[4] M. A. Ayu, T. Mantoro, A. F. A. Matin, and S. S. Basamh. Recognizing user activity based on accelerometer data from a mobile phone. In *Computers & Informatics (ISCI), 2011 IEEE Symposium on*, pages 617–621. IEEE, 2011.

[5] H.-Y. Chen, Y.-H. Lin, and C.-M. Cheng. Coca: Computation offload to clouds using aop. In *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, pages 466–473. IEEE, 2012.

[6] D. Chu, N. D. Lane, T. T.-T. Lai, C. Pang, X. Meng, Q. Guo, F. Li, and F. Zhao. Balancing energy, latency and accuracy for mobile sensor data classification. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, pages 54–67. ACM, 2011.

[7] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 49–62. ACM, 2010.

[8] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[9] S. Kaghyan and H. Sarukhanyan. Activity recognition using k-nearest neighbor algorithm on smartphone with tri-axial accelerometer. *International Journal of Informatics Models and Analysis (IJIMA), ITHEA International Scientific Society, Bulgaria*, 1:146–156, 2012.

[10] M. Kose, O. D. Incel, and C. Ersoy. Online human activity recognition on smart phones. In *Workshop on Mobile Sensing: From Smartphones and Wearables to Big Data*, pages 11–15, 2012.

[11] J. R. Kwapisz, G. M. Weiss, and S. A. Moore. Activity recognition using cell phone accelerometers. *ACM SigKDD Explorations Newsletter*, 12(2):74–82, 2011.

[12] S. Lee. *A framework for runtime energy efficient mobile execution.* PhD thesis, University of Southern California, 2016.

[13] K. Naik. *A survey of software based energy saving methodologies for handheld wireless communication devices.* Department of Electrical and Computer Engineering, University of Waterloo, 2010.

[14] M.-R. Ra, B. Priyantha, A. Kansal, and J. Liu. Improving energy efficiency of personal sensing applications with heterogeneous multi-processors. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, pages 1–10. ACM, 2012.

[15] M. Shoaib, H. Scholten, and P. J. Havinga. Towards physical activity recognition using smartphone sensors. In *Ubiquitous Intelligence and Computing, 2013 IEEE 10th International Conference on and 10th International Conference on Autonomic and Trusted Computing (UIC/ATC)*, pages 80–87. IEEE, 2013.

[16] M. Shoaib, S. Bosch, O. D. Incel, H. Scholten, and P. J. Havinga. Fusion of smartphone motion sensors for physical activity recognition. *Sensors*, 14(6):10146–10176, 2014.

[17] Y. Wang, J. Lin, M. Annavaram, Q. A. Jacobson, J. Hong, B. Krishnamachari, and N. Sadeh. A framework of energy efficient mobile sensing for automatic user state recognition. In *Proceedings of the 7th international conference on Mobile systems, applications, and services*, pages 179–192. ACM, 2009.

[18] Z. Yan, V. Subbaraju, D. Chakraborty, A. Misra, and K. Aberer. Energy-efficient continuous activity recognition on mobile phones: An activity-adaptive approach. In *2012 16th international symposium on wearable computers*, pages 17–24. Ieee, 2012.