Asian Journal of Information Technology 15 (13): 2177-2185, 2016
ISSN: 1682-3915

Context Aware Offloading Decision and Partitioning in Mobile Cloud Computing

[1]N.M. Dhanya and [2]G. Kousalya
[1]Department of Computer Science and Engineering, Amrita School of Engineering,
Amrita Vishwa Vidyapeetham, Amrita University, 641112 Coimbatore, India
[2]Department of Computer Science, Coimbatore Institute of Technology, 641014 Coimbatore, India

Abstract: The development of cloud and mobile technology leads to Mobile Cloud Computing (MCC). MCC has become a major service structure now a days. The limitations in the battery power of mobile can be overcome with the help of cloud technology which is having infinite amount of resources. Offloading is a method for improving the capabilities of resource limited smartphones by augmenting with cloud resources. The mobile applications can be partitioned into two in such a way that heavier parts are executed at the cloud and the rest is executed in the mobile itself. This study designs a system for offloading and partitioning architecture which will take into consideration of all the contextual information related to a mobile. The decision of offloading and partitioning is taken considering the current connectivity, memory status, battery charge, etc. The evaluation results reveal that this algorithm gives performance improvement, less overhead to the mobile side and the prediction accuracy of context aware decision engine. A light weight partition algorithm is used for splitting the application. The results shows significant improvement in time and energy consumed.

Key words: Mobile cloud, context aware offloading, decision making, application partitioning, offloading

INTRODUCTION

Cloud computing and mobile technologies are getting increasingly popular. Mobile devices such as smartphones and tablets are becoming more and are more powerful and always connected to the internet with the technologies like Wi-Fi/3G/4G. Even though, it is always connected, the major limitation the mobile devices are facing is stringent resource constraints such as memory, battery lifetime, CPU power compared to the laptop and desktop counterpart (Liu *et al*., 2013; Dinh *et al*., 2013). Among all the above resources, the battery power is the most critical one to be considered. If an application is to be performed which is too resource intensive such as a gaming application, the battery charge will drain off quickly which may lead to poor Quality of Experience (QoE).

The cloud can be used as an effective way of augmenting the mobile for better performance. Offloading is a technique through which the resource intensive tasks can be migrated to the cloud for execution and the results can be collected back to the mobile. But offloading is not always beneficial. If the bandwidth is too low, it will take a long time for the data transfer between the cloud and the mobile and it will take a long time for completing the offloading tasks than on the mobile itself. Hence, the decision of offloading should be taken in such a way that offloading will lead to significant performance improvement. This study is dealing with a method which is considering all contextual information for an efficient offloading decision making. Another contribution is the application partitioning algorithm which will partition the application into two, a local one and a remote one for executing in the mobile and the cloud respectively. This decision also considers the disconnection factor.

Various studies have performed on how to make "longer battery lifetime". The resource limitation can be improved by offloading the application to a resource rich servers on the cloud making Mobile Cloud Computing (MCC) (Kumar and Lu, 2010). In recent years, a number of research has been taking place for offloading decision making (Wolski *et al*., 2008; Li *et al*., 2015) and application partitioning (Liu *et al*., 2015). The main factors to be considered in offloading are the characteristics of the mobile device, the wireless communication link properties, the application demands and the user requirements. The heterogeneous nature of the above factors make the offloading a complex process to do. Even though enough benefits are there on computational offloading, offloading is beneficial only if the saved computational time on mobile is less than the computational delay in the clouds. Most of the previous research makes the decision and

Corresponding Author: N.M Dhanya, Department of Computer Science and Engineering, Amrita School of Engineering, Amrita Vishwa Vidyapeetham, Amrita University, 641112 Coimbatore, India

partitioning engine in the mobile itself which will reduce the efficiency of the offloading algorithm because of the additional burden of this decision and partitioning. In our research, we are shifting all these functionalities to the cloud so that no additional research is there in the resource constrained mobile devices.

The contribution of our research can be summarized as follows. We are introducing an architecture where a light weight decision and partitioning algorithms are selected to reduce the burden of the mobile devices. We propose context aware algorithms so that the decision is based on the current context. A partitioning algorithm which partitions the application into local and remote for execution.

Literature review: Recently, a number of researches are going in the direction of offloading decision making and application partitioning. Niu *et al.* (2014) proposed a bandwidth adaptive partitioning for distributed execution optimization of mobile applications approach which is taking into consideration of the bandwidth changes and making the partitions accordingly. A bandwidth adaptive algorithm is proposed for this. This study uses static partitioning and dynamic profiling for decision making. Zhang *et al.* (2012) proposed a partitioning algorithm with high accuracy. It is based on a call graph strategy and finds a partitioning with minimum cost and minimum time complexity. The code partitioning algorithm uses a DFS and liner time searching algorithm.

Magurawalage *et al.* (2014) proposes an energy efficient and network aware offloading algorithm for mobile cloud computing. This research is based on a middleware cloudlet layer. The mobile device communicate with the cloud using cellular network. Because of the mobility there is a chance of network disconnection. If the connection to the cloudlet is lost there is a provision to connect to a different cloudlet. There is enough mechanism for data recovery too in case of disconnection.

Lin *et al.* (2015) research is based on Time and energy aware computation offloading in handheld devices to coprocessors and clouds. This study is considering both time and energy while offloading which reduces the response time and reduces energy consumption at the same time. It is dealing with a ternary decision engine for making the offloading decision.

Barbera and coauthors proposed a bandwidth and energy costs of mobile cloud computing by is dealing with mobile computation offloading in real life scenarios. They consider two types of clones: the off-clone whose intention is to support computation offloading and the back-clone which comes to use when a restore of user's data and apps is needed. They attain this through measurements done on a real test bed of 11 Android smartphones and an equal number of software clones running on the Amazon EC2 public cloud. Verbelen *et al.* (2013) proposes a graph partitioning algorithms for optimizing software deployment in mobile cloud computing. This study is dealing with software partitioning for offloading to the cloud. The proposed approach is minimizing the bandwidth requirements. They are using a KL based refinement for graph partitioning.

Ellouze *et al.* (2015) offloading architecture is one of the latest research for offloading. This study considers the current CPU load and State of Charge (SoC) of the mobile phones as the parameters for offloading. A decision is taken for offloading based on the delay for offloading and Quality of Experience (QoE). The decision parameters are the critical delay, energy balance and the state of charge of battery. A chess game, speech recognition and a virus scan application is considered and the results are compared based on the energy gain and rejection ratio.

In summary, there are lot of interesting approaches to various issues in offloading. However, all are dealing with the calculations in the mobile which will make the energy issue more critical. Comparing to the existing research, the aim of our approach is to move all the calculation related to the offloading decision making and partitioning to the cloud so that no extra burden is given to the phones and efficiently managing the power. The partitioning algorithm which is presented in this study considers the network disconnection factor which will increase the efficiency of partitioning in dynamic environment like mobile.

MATERIALS AND METHODS

System architecture and problem formulation

Architecture: The architecture for decision making and partitioning is as shown in Fig. 1. Once, the application starts running the context information is collected by the profiler module. The profiler module will collect all the context information from the mobile and the network and it will be stored in the context database. From the data in the context database, a decision is taken whether to offload the application to the cloud or not. If the decision is "yes", the partitioning module will take care of the application partitioning and scheduling. The mobile device will be informed of the offloading and partitioning decision and only those modules which are referred to as local will be executed in the mobile, rest of the applications modules will be executed on the cloud and

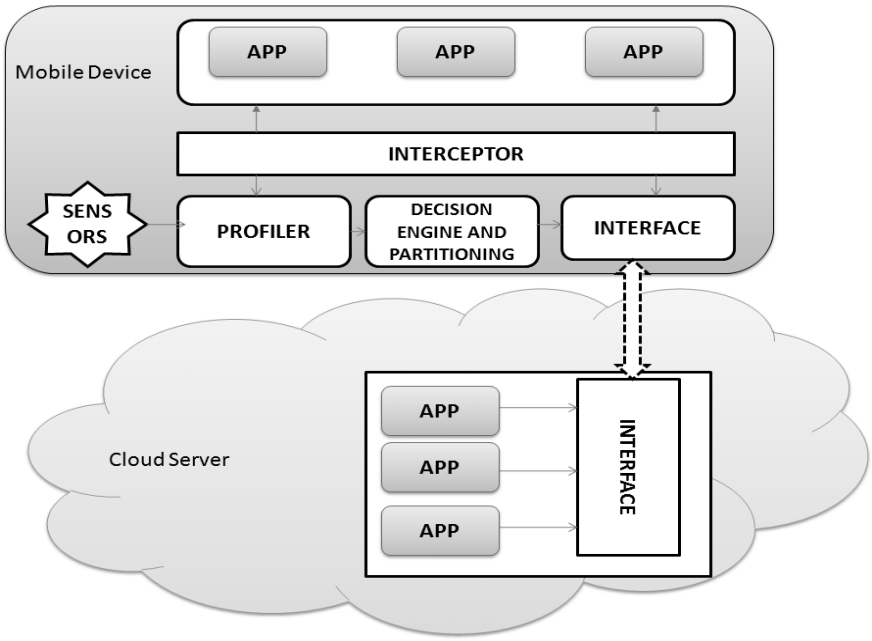

Fig. 1: Architecture

the results are send back to the mobile. The interceptor module is responsible for passing the parameter list, return type, etc. If the offloading decision is yes and after partitioning, it serializes all the parameters and sends to the interface. A corresponding interface at the cloud server receives all these parameters and pass this to the application part. After the execution through the same interface the results are send back to the device. The interface at the mobile side deserializes the result to build the appropriate object and returns to the invoker of the method. In case of a local execution or a remote server failure, this architecture lets the application continue its normal execution flow.

If an offloading decision and partitioning is made, all the input parameters are collected for the method to be offloaded and pack them into a byte array. The name of the application, method, class, parameters to the methods, etc are collected and packed. Once, this is ready the interceptor will send this to the interface for passing to the cloud server. The interface communicates with the cloud server to execute the application part with the input parameters and gets the result back to the interceptor. The interceptor gives this back to the invoked application and method.

The profiler module dynamically collect the system parameters and resource information such as network bandwidth, latency, memory, CPU load both at mobile and the cloud. The latency is calculated by sending an ICMP packet to the cloud server. The CPU and memory availability is calculated by examining proc/stat and proc/meminfo files of the android and the cloud server. The data size is varied in the application. For example, for the sorting application which is created, different array sizes are given to simulate different data size.

On the cloud side, the same application is installed in a virtual machine, so that, once a call is coming the same method can be called at the server side. The cloud receives the offloading request along with the parameters through the interface. This initiates the corresponding methods at run time from the provided input parameters, executes the method through the application interface and after completion the results are send back to the device through the interface.

Profiler: This is one of the main function of the offloading prediction framework. The profiler will collect all the context information from the mobile and the cloud and store it is a context database. From the data in this dataset a decision of offloading is taken. The profiler is a background service which constantly collects the following information:

- Signal strength (B): The profiler periodically the collects the bandwidth (signal strength) of Wi-Fi or 3G
- Data size to be transmitted (D)
- Free Memory status of the mobile in percentage (M_{memory})
- CPU load on the mobile in percentage (M_{cpu})

Table 1: Measures of the applications

Parameters	Loop	Sorting	Face detection	Face recognition
FP	3	13	7	10
FN	5	15	6	9
TP	556	672	549	689
TN	452	356	464	330
Sensitivity	0.991087	0.978166	0.989189	0.987106
specificity	0.993407	0.96477	0.985138	0.970588
Accuracy	0.992126	0.973485	0.987329	0.981696
Precision	0.994633	0.981022	0.98741	0.985694
NPV	0.989059	0.959569	0.987234	0.973451
F-measure	0.992857	0.979592	0.988299	0.986399
Misclassification rate	0.007874	0.026515	0.012671	0.018304

- Battery Charge (Batt)
- Load at the cloud (C_{cpu})

The applications considered can be divided into four:

- Low Computation with Small Data transfer (LCSD), e.g., simple looping applications
- Low Computation with Large Data transfer (LCLD), e.g., nested loop with multiplication and addition or sorting an array of random numbers 1000 times
- High Computation with Small Data transfer (HCSD), e.g., face detection or simple color space conversion from RGB to YUV
- High Computation with Large Data transfer (HCLD), e.g., face recognition

All applications in real world will be coming under any of the four category. Android applications are developed for collecting the sample dataset. For simple loops and sorting the cloud part is implemented in the google app engine and face detection and recognition application a tomcat server is setup as cloud. Each application considered is executed 1000 times to find the local and remote execution time. If the local execution time is less the application is marked as local (0) and if the remote execution time is less, the application is marked as remote. The data set is created in this manner and is used as the training set for the classification. The sample dataset is given in Table 1 for sorting application. The data field represents the number of elements to be sorted.

Offloading decision engine: The main challenge associated with an offloading model include whether the offloading is beneficial and how to do offloading if it is beneficial. A decision should be taken for offloading first and then determine which portions to be offloaded. However, existing research addressing these challenges have some of the following limitations:

- Offloading is beneficial only if it takes less time and energy for execution and it should consider all dynamic factors which affects offloading. Always offloading policy will deteriorate the system if the context is not favorable
- Linear regression model is a common method used which can consider only limited number of features
- The parameters affecting the offloading decision are highly dynamic

Considering all these limitations offloading decision can be considered as a classification problem which is highly dynamic, low biased and tolerant to the noise. Because this researchs on the mobile, the classifier should be light weight, adaptive and should be self learning gradually overtime and consumes less energy and time.

The decision algorithm assesses a processing task and determines the location for execution that would maximize the energy conservation and minimizes the execution time on the mobile device. The trade-offs between local and remote execution influence the decision made. In other words if local computation cost is larger than communication cost, offloading is advised and vice versa.

The decision engine selects the most energy saving task execution option. For this, the context data is parsed and processed for fine tuning the decision of local or remote execution. Following parameters are considered for decision making which is obtained from the context database:

$$<B,D,M_{memory}, M_{cpu},Batt, C_{cpu}>$$

By comparing different classification algorithm the conclusion is given as Fig. 2. The time, energy and accuracy varies slightly. Even though SVM and MLP has higher accuracy ratio, the time and energy consumed for regression is less. This study the classification technique used is Multiple logistic regression. SVM is having high computational burden which is not desirable for this system because time and energy are constraints for our mobile device. This disadvantage can be overcome by LS-SVM which solves linear equations rather than quadratic programming. The independent variables are the

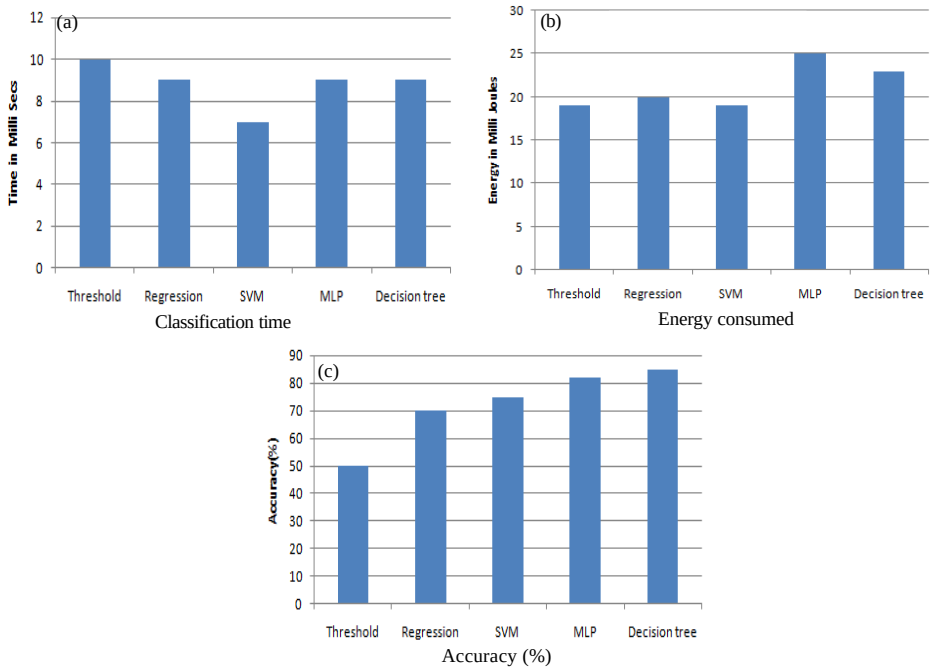

Fig. 2: a-c) Comparison between different classifiers

above tuple from the data set and the dependent variable is the decision of local/remote execution. Since, the logistic regression takes categorical data, it is suitable for the offloading decision. The two categories of decision are Local or remote execution.

Another advantage of this regression analysis is it is simple and less time and energy consuming. From the previous data which is available in the data set, the classifier will predict the fate of the current execution. Figure 2a-c shows the energy, time and accuracy for regression analysis compared to other classification and prediction techniques. Compared to Threshold, SVM (Support Vector Machine), MLP (Multilayer Perceptron) and decision tree regression takes less time and less energy. But the classification accuracy is slightly lower than some of the methods. Because this should be executed in the resource limited smartphones the considerations are only time and energy as the major factor and selected regression analysis for classification.

Based on collected information in the dataset , it makes the decision either to offload the method or to execute it locally. The classifier has a feedback channel as well so that it can learn by itself through the entire decision process. The following shows the measures for the logistic regression analysis of the decision making.

Partitioning module: For partitioning the application is converted into a Weighted Object Relational Graph (WORG). If the decision taken in the decision or classifier module is "yes" then a partitioning algorithm will take care of the application division and the parts which is classified as cloud will be executed in the cloud and local will be executed in the smartphones.

In this method, the mobile application which is to be offloaded is partitioned into two for local and remote execution. First of all the application should be converted into WORG. The partition strategy makes minimum data transfer. So that the application is executed in a distributed manner with minimum data transfer cost.

Weighted Object Relational Graph (WORG): The application is converted into weighted object relational graph using Soot Analysis frame work. Figure 3 shows a sample WORG of a face recognition application. Each class is represented as a circle node. An edge connecting two nodes represents the communication between the nodes corresponding to the data size which are to be transferred. A static analysis is done on the source code to make the WORG. The example shows four classes, face preview, Image capture, face detection and face detection library, where the face preview is the main class and from that image capture and face detection are called. The edge weights represent the communication weights between the nodes.

Application partitioning algorithm: Let a graph G = (V, E) be the application graph where G is the vertex and E is the

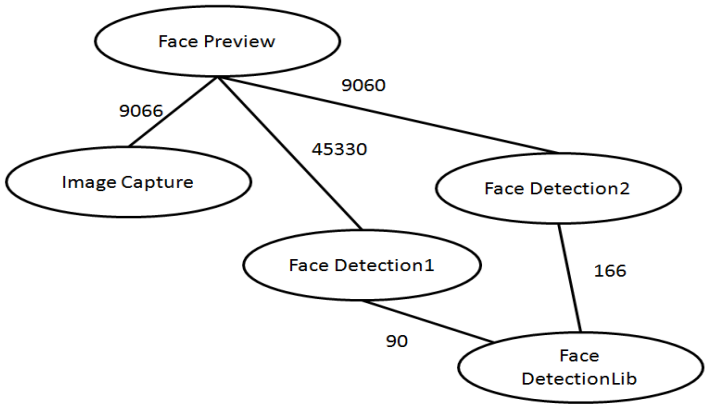

Fig. 3: WORG

edges between the vertices. Let be the total number of vertices. The vertex set can be represented as $v = \{v_i\text{-}||i\epsilon[1, 2, ..., m]\}$. The edge set can be represented as $E = \{C_{ij}\text{-}||i, j\epsilon[1, 2, ..., m]\}$ where C_{ij} represents the communication cost between node V_i and $V_{.j}$ The communication cost can be represented as the data transferred between node V_i and V_j while executing. Not all vertex can be offloaded. Some of the vertices are device dependent such as user interface, sensor related classes, etc. Such vertices can be marked as non-movable prior to execution. Let n be the number of vertices that can be moved where $n \leq m$. Let A and B is a solution at a point where the vertices are divided into $<V_{mobile}, V_{cloud}>$ where V_{mobile} represents the nodes at the mobile side and V_{cloud} represents nodes at the cloud side. The conditions applied are there is no overlapping between the mobile side and cloud side nodes and no nodes are left free (all nodes should be include in any one of the partitions). $V_{mobile} \cap V_{cloud}$ and $V_{mobile} \cup V_{cloud} = V$.

Algorithm for application partitioning: A graph partitioning algorithm is used to split the graph into local and remote partitions. A graph splitting algorithm Kernighan-Lin is used to bi-partition the graph. The cost which are used for partitioning is the edge weight of the WORG which represents the communication cost between two nodes. The partitioning is done in such a way that there will be minimum communication between the cloud and the mobile. Since KL is one of the simplest graph partitioning algorithm, our resource constraint Smartphone can adapt this algorithm for partitioning the applications.

Algorithm A; Graph partitioning algorithm:

Compute T = cost (A, B) of initial partition A, B
Repeat
Compute costs D(n) for all n in N
Unmark all nodes in G
While there are unmarked nodes
Find an unmarked pair (a, b) maximizing gain (a, b)
Mark a and b (but do not swap them)
Update D(n) for all unmarked n as though a and b had been swapped
End while
Pick j maximizing Gain = $sum_{i=1..j}$ gain (i)
If Gain >0 then
Update A = A-{a_1, ..., a_j} U {b_1, ..., b_j}
Update B = B-{b_1, ..., b_j} U {a_1, ..., a_j}
Update T = T-Gain
End if
Until Gain <=0

RESULTS AND DISCUSSION

The implementation of the decision making and partitioning algorithms uses a Gionee V4S Smartphone (Low end) and Samsung Galaxy S4 (high end). The server is installed on apache tomcat and Google app engine for cloud for real time applications. The applications are randomly executed and the time and energy are calculated. Power tutor is used for measuring the power consumption of the applications.

Real time application results: A Real time android Application is developed for this experiment. This application was using randomly by 5 members of the University and the results are based on the average value obtained from all these users. The applications taken into consideration are simple looping, sorting, face detection and face recognition (Table 2).

The following figures shows the performance of our approach compared to the no offloading and total offloading scenario. Even though total offloading gives almost equal performance as context aware for high bandwidth, in the case of low bandwidth total offloading

Table 2: Sample dataset generated

Mobile CPU load (%) [M_{CPU}]	Battery charge (%) [Batt]	Free memory on mobile [M_{memory}]	Bandwith [B]	Data [1-1000] [D]	Cloud load (%) [C_{CPU}]	Execution
11	41	905	23	123	58	Remote
33	97	503	74	1000	56	Remote
85	42	276	52	606	51	Remote
2	18	648	30	761	92	Local
10	30	137	39	826	44	Local
3	16	644	94	809	94	Local
5	38	711	96	975	44	Local
9	52	994	55	339	71	Remote
89	78	270	27	119	33	Remote
11	50	438	88	102	94	Remote
43	10	934	53	917	15	Remote
61	57	546	58	346	85	Remote
28	87	415	67	111	23	Remote
6	23	228	75	145	63	Remote
19	95	536	73	277	25	Remote
60	70	195	59	977	91	Remote
20	42	311	26	790	12	Remote
8	94	674	77	549	78	Remote
6	33	395	76	749	75	Local
76	73	275	53	956	62	Remote

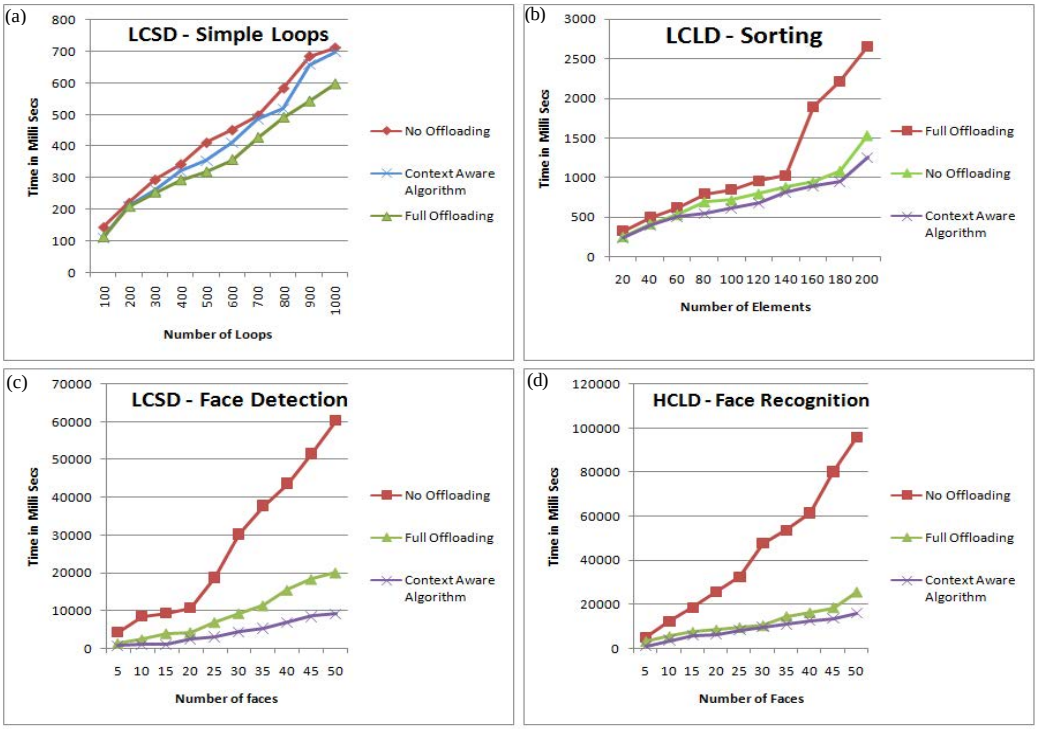

Fig. 4: Execution time

gives poor performance than context aware offloading. Figure 4 and 5 shows the preliminary results. Compared to total offloading, the user's response time with Context Aware approach is reduced to 59.39, 9, 11.99 and 72.28% of the response time if face detection, sorting, looping, face recognition applications are always offloaded to the server. Correspondingly, in Context Aware approach, the energy consumed on the Smartphone is 57.21, 9.49, 25.80 and 90.01% of the energy consumed if these applications are always offloaded.

On the other hand, compared to No offloading where the applications are always executed locally, Context aware approach also provides the user faster response time for face detection and face recognition (34.70 and

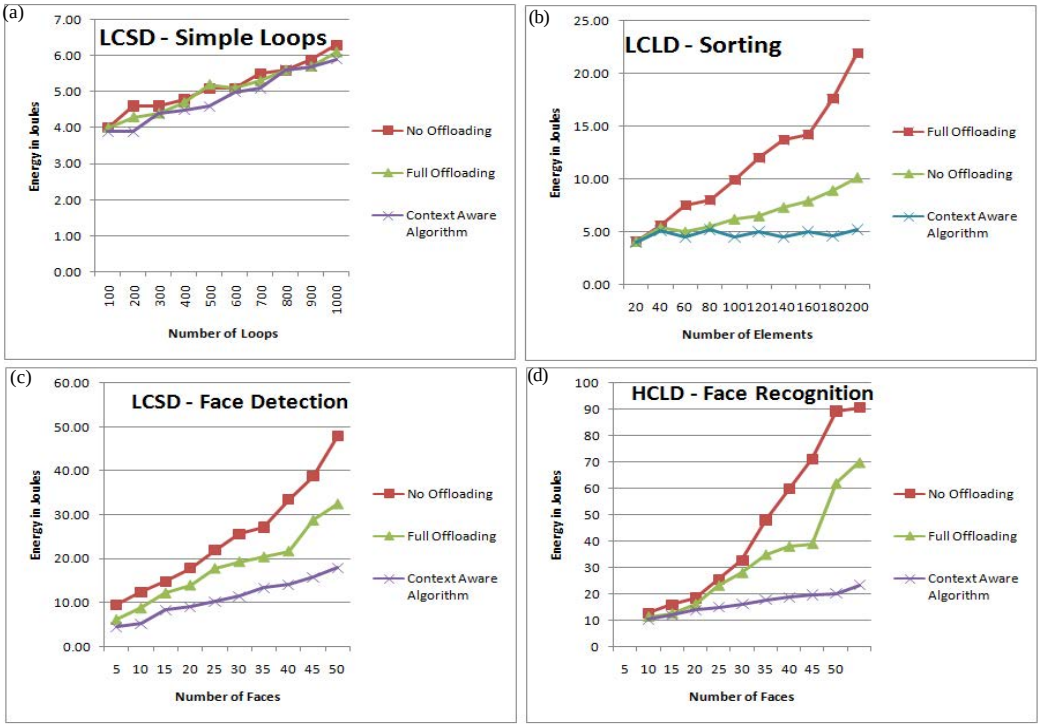

Fig. 5: Execution energy

81.32% of their local execution time, respectively). The corresponding energy consumption is about 48.61 and 89.48%, respectively of the energy consumption when they are executed locally. For looping, context aware approach's performance is similar to that of the local execution and sorting for lower bandwidth local execution is better than remote execution because of large data transfer.

Among the evaluated applications, the context aware approach does not perform well for looping and sorting. This is because sorting is purely data intensive and always has the best performance when executing on device. Looping is neither data-nor computation-intensive. Looking into the applications, the execution time on the device is very small. Thus training and classification time in context aware approach becomes significant overhead compared to its actual execution time (in terms of percentage). In general, data intensive tasks will never get the benefits of offloading because of the high data transfer between the cloud and the mobile.

CONCLUSION

The proposed architecture where the offloading is based on the current context of the mobile device. In the cloud the profiler is dealing with the collection of all the context information. This will be given to the decision maker for making the decision of offloading. With the help of the energy model the decision is taken whether to offload or not. If the decision is to offload the control is given to the partitioning module. The partitioning module with the help of the information from the profiler partitions the applications and inform the mobile devices about the partition and execution starts. With the numeric results obtained there is improvement in the energy and time of execution with this method. Since the calculations are totally done at the cloud side more improvements are showing in the results compared to the existing methods. In future more applications can be taken into consideration. More real time applications can be used to verify the results.

REFERENCES

Dinh, H.T., C. Lee, D. Niyato and P. Wang, 2013. A survey of mobile cloud computing: architecture, applications and approaches. Wirel. Commun. Mobile Comput., 13: 1587-1611.

Ellouze, A., M. Gagnaire and A. Haddad, 2015. A mobile application offloading algorithm for mobile cloud computing. Proceedings of the 3rd IEEE International Conference on Mobile Cloud Computing, Services and Engineering, March 30-April 3, 2015, IEEE, New Orleans, Louisiana, pp: 34-40.

Kumar, K. and Y.H. Lu, 2010. Cloud computing for mobile users: Can offloading computation save energy?. Comput., 43: 51-56.

Lin, Y.D., E.T.H. Chu, Y.C. Lai and T.J. Huang, 2015. Time-and-energy-aware computation offloading in handheld devices to coprocessors and clouds. IEEE. Syst. J., 9: 393-405.

Liu, F., P. Shu, H. Jin, L. Ding, J. Yu, D. Niu and B. Li, 2013. Gearing resource-poor mobile devices with powerful clouds: Architectures, challenges and applications. IEEE Wireless Commun., 2: 14-22.

Liu, J., E. Ahmed, M. Shiraz, A. Gani and R. Buyya *et al*., 2015. Application partitioning algorithms in mobile computing: Taxonomy, review and future directions. J. Network Comput. Appl., 48: 99-117.

Magurawalage, C.M.S., K. Yang, L. H and J. Zhang, 2014. Energy-efficient and network-aware offloading algorithm for mobile cloud computing. Comput. Networks, 74: 22-33.

Niu, J., W. Song and M. Atiquzzaman, 2014. Bandwidth-adaptive partitioning for distributed execution optimization of mobile applications. J. Network Comput. Appl., 37: 334-347.

Verbelen, T., T. Stevens, F.D. Turck and B. Dhoedt, 2013.Graph partitioning algorithms for optimizing software deployment in mobile cloud computing. Future Generation Comput. Syst., 29: 451-459.

Wolski, R., S. Gurun, C. Krintz and D. Nurmi, 2008. Using bandwidth data to make computation offloading decisions. Proceedings of the IEEE International Symposium on Parallel and Distributed Processing, April 14-18, 2008, IEEE, Miami, Florida, ISBN: 978-1-4244-1694-3, pp: 1-8.

Zhang, Y., H. Liu, L. Jiao and X. Fu, 2012. To offload or not to offload: an efficient code partition algorithm for mobile cloud computing. Proceedings of the 1st International Conference on Cloud Networking, November 28-30, 2012, IEEE, Paris, France, ISBN: 978-1-4673-2797-8, pp: 80-86.