# Machine Learning-based Runtime Scheduler for Mobile Offloading Framework

Heungsik Eom, Pierre St Juste, Renato Figueiredo
*Advanced Computing and Information Systems Laboratory*
*Electrical and Computer Engineering*
*University of Florida, Gainesville, Florida, USA*
{*hseom, pstjuste, renato*}*@acis.ufl.edu*

Omesh Tickoo, Ramesh Illikkal, Ravishankar Iyer
*Intel Corporation*
*2111 N.E. 25th Avenue*
*Hillsboro, Oregon, USA*
{*omesh.tickoo, ramesh.g.illikkal, ravishankar.iyer*}*@intel.com*

***Abstract*—Remote offloading techniques have been proposed to overcome the limited resources of mobile platforms by leveraging external powerful resources such as personal workstations or cloud servers. Prior studies have primarily focused on core mechanisms for offloading. Yet, adaptive scheduling in such systems is important because offloading effectiveness can be influenced by varying network conditions, workload requirements, and load at the target device. In this paper, we present a study on the feasibility of applying machine learning techniques to address the adaptive scheduling problem in mobile offloading framework. The study considers 19 different machine learning algorithms and four workloads, with a dataset obtained through the deployment of an Android-based remote offloading framework prototype on actual mobile and cloud resources. From this set, a subset of machine learning algorithms, which have relatively high scheduling accuracy, is selected to implement an offline offloading scheduler. Finally, by taking computational cost and the scheduling performance into account, we use Instance-Based Learning to evaluate an online adaptive scheduler for mobile offloading. In our evaluation, we observe that an Instance Learning-based online offloading scheduler selects the best scheduling decision in 87.5% instances, in an experiment setup in which an image processing workload is offloaded while subject to varying network bandwidth conditions and the amount of data transfer.**

***Keywords*-Mobile platform, cloud, offloading, machine learning, scheduling, energy consumption**

## I. Introduction

Rapid enhancements in computing capabilities of mobile platforms have been driving the increased adopting and use of mobile computing platforms by increasing numbers of users. Today's mobile platforms are able to deliver capabilities that are close to those of non-mobile platforms such as desktops or workstations. For instance, a mobile phone equipped with a Graphic Processing Unit (GPU) core is able to achieve approximately 10GFLOPS/Watt of computer-power, which is identical as a 4-core desktop with GPU [1]. Despite of these significant advancements, mobile platforms remain significantly limited by resources such as memory size, storage capacity, and especially battery lifespan. To alleviate the problem of the resource limitations in mobile platforms, computation offloading techniques have been proposed as a way to extend the capabilities of mobile platforms to more powerful resources. These may include personal computers, servers, or even public cloud resource over the network [2], [3], [4].

However, the benefits from these systems can vary due to different requirements for data transfer among various types of mobile applications and dynamic network conditions including latency and bandwidth. As a result, offloading is not always beneficial, and poor offloading decisions can result in the degradation of performance or energy consumption. Therefore, offloading frameworks need to consider the scheduling of workloads onto remote or local processing resources adaptively, as a function of network conditions and application requirements.

In this paper, we address these challenges by considering machine learning techniques for a runtime adaptive scheduler for mobile offloading framework. Machine learning technique is a branch of artificial intelligence through which a system can learn from previous data and adapt to unseen situations dynamically. By applying machine learning techniques to the remote offloading scheduling problem, a scheduler can be automatically trained from previous offloading behaviors and make decisions on whether the mobile workload should be offloaded or executed locally informed by past behavior and current conditions. There have been a number of related studies proposing adaptive offloading mechanisms for mobile platforms. To the best of our knowledge, our work is the first to systematically study machine learning techniques to a mobile offloading scheduler. To this end, we have developed a remote offloading system based on the OpenCL framework, which is the well-defined hardware-level offloading API [5]. With the OpenCL-based remote offloading framework, we perform detailed measurement experiments under various network conditions and mobile applications to show the necessity and efficacy of an adaptive offloading mechanism. One of the contributions of our work is the combination of dynamic network conditions and application requirements into *Computation-to-Communication ratio*, which considers the local processing time for the workload, the amount of data transfer, and network bandwidth. The computation-to-communication ratio is a composite measurement which coalesces three dynamic features into one parameter, and is used as an attribute of the machine learning technique.

CFP13UCC-USB/13 $26.00 © 2013 IEEE
DOI 10.1109/UCC.2013.21

For the evaluation on the feasibility of applying machine learning techniques to the adaptive scheduling problem, we utilized *Weka* [6], a Java-based open source package.

After investigating the scheduling accuracy of several machine learning algorithms using Weka, we choose a few machine learning algorithms which have relatively high scheduling accuracy to implement an *offline* offloading scheduler. Further, by taking the complexity and scheduling performance into account, we select Instance-Based Learning algorithm for an *online* scheduler for mobile offloading framework. In the evaluation, we show that although Instance-Based Learning online offloading scheduler is fairly simple and has low overhead, it provides performance advantages over non-adaptive schedulers for mobile offloading.

The rest of the paper is organized as follows. In Section II, we overview previous works on scheduling problems in mobile offloading frameworks, as well as machine learning techniques for dynamic runtime schedulers. Section III discusses the challenges in remote offloading scheduling based on empirical and experimental data. Section IV describes various machine learning techniques used for mobile offloading framework. In Section V and VI, we evaluate various offline machine learning-based runtime schedulers and show the potential benefits of the implementation of online runtime scheduler. Finally, we conclude the paper in Section VII.

## II. Related work

### A. Adaptive Mobile Offloading

Many studies have considered adaptive mobile offloading to provide performance improvements and energy savings for mobile platforms. In [7], the authors focus on relieving the memory limitation of a mobile device by dynamically making the offloading decision with Offloading Inference Engine (OLIE) which is based on the fuzzy control model. In particular, OLIE profiles the available memory size of a mobile device and network bandwidth, and maps them into the offloading decision specifications by the application developer, such that when the current condition matches any specified rule, an offloading action is triggered. In MAUI [3], the authors assume that offloading is always preferable to local processing; however, it depends on three factors to determine which methods should be offloaded to the remote server: the device's energy consumption characteristics, the program characteristics, and the network characteristics. Specifically, MAUI used the lightweight throughput measurement to profile network condition. In [8], a prediction model for the performance of distributed mobile applications is evaluated through a sample image processing application (i.e. face detection). The prediction model heuristic uses linear functions to approximate the time for local, remote execution and data transfer. The server updates these functions using least-squares method, and returns the updated heuristic linear functions to the client so that those updated functions are used for the performance comparison between local processing and offloading. Kovachev et al. [9] propose a simple cost function of a service-based mobile cloud computing middleware for Android platforms under three restrictions: minimized memory usage, minimized energy usage, and minimized execution time.

Mobile "Grid" systems, where mobile devices participate as resource users or providers, also consider scheduling techniques to improve the performance of the systems while tackling the resource constraints of mobile platforms. In [10], a novel energy-aware scheduling is formulated and the level-based list scheduling heuristic is proposed for a Mobile wireless Ad hoc NETwork (MANET). The authors predefine the models for the task, the processor or the mobile device, and the cost function, and the scheduler tries to minimize the cost function by mapping each task to the resource through the level-based list scheduling heuristic.

Although the above-mentioned approaches take into account dynamic parameters from the application level or the network level (i.e. CPU cycles, network bandwidth) to predict system performance and schedule the mobile workload execution, they still rely on predefined decision rules or cost models, preventing the scheduler from adapting to dynamic conditions during runtime. In contrast, in this paper we consider approaches that do not rely on any predefined specifications or prior knowledge of the mobile application. Instead, we consider machine learning techniques for adaptive runtime mobile offloading schedulers. Once trained with training data, the scheduler predicts the behavior of an incoming task when deciding on local or remote execution.

### B. Machine Learning Techniques for Dynamic Schedulers

Machine learning techniques have been used to address dynamic scheduling problems in various areas, such as heterogeneous computing platforms, grid computing systems as well as in data center. In [11], machine learning techniques are used to provide a compiler-based, automatic and portable predictor for multi-core processors. In order to determine the best number of threads and scheduling policy, the authors used a feed-forward artificial neural network and a multi-class support vector machine model, respectively. Berral et al. [12] propose an energy-aware data center through server consolidation by turning off idle servers with assistance from machine learning based scheduling. The scheduler predicts the future performance of the jobs and power consumption in the resulting job allocation using linear regression algorithms. The novel Adaptively Scheduled parallel R (ASpR) framework, which transparently parallelizes scripts in the popular R language, is presented in [13]. This framework uses artificial neural networks for the performance modeler which predicts task computation and data communication costs, and this modeler is used by the directed acyclic graph to determine an appropriate schedule.

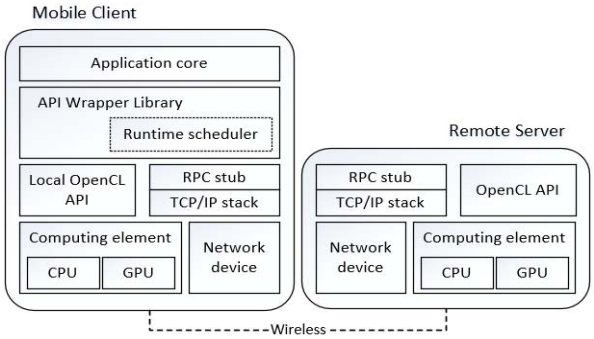



Figure 1. Overall architecture of the OpenCL-based remote offloading framework for mobile platforms

In our work, we further consider the appropriateness of adopting machine learning techniques to the scheduler of our mobile offloading framework with respect to the complexity to construct the predictor and ability to capture dynamic characteristics of mobile environments.

## III. Adaptive scheduling challenge for remote offloading framework

In this section, we describe the baseline mobile offloading framework on which we build the runtime scheduler based on machine learning techniques. Then, we conduct experiments to highlight the potential advantages of adaptive offloading mechanisms.

### A. Mobile Offloading Framework

OpenCL is the open standard for parallel programming of heterogeneous systems, which are increasingly found in personal computers, servers and mobile devices. By offloading computations to more powerful computing elements such as Graphics Processing Units (GPUs) or special hardware accelerators (e.g. SSL accelerators or FPGA) at the hardware layer, it is possible to improve the performance for a wide range of applications from gaming and entertainment to scientific and medical software [5]. The offloading mechanism considered in this study leverages the OpenCL API to support the remote offloading over the network. As such, the framework inherits the ability of heterogeneous computing of the OpenCL standard. The key idea of the OpenCL-based remote offloading framework (Figure 1) is to integrate the OpenCL API with an RPC-based service through an API wrapper library which has an identical name and signature as the original OpenCL API.

When an application invokes an OpenCL API, the API wrapper library captures this API call, and a runtime scheduler makes a decision on offloading or local execution. If the scheduler decides to offload the call, it marshalls arguments for the API and invokes an RPC call associated with the API. Finally, a function is executed on the remote server, and the result is sent back to the mobile client. On the other hand, if the function should be executed locally, the wrapper library calls the local OpenCL API and the execution result is returned to the application directly.

For an RPC service, we have developed a light-weight marshalling and remote procedure call layer for our offloading framework. By running our own RPC-based service, we provide a workload offloading design that is efficient in terms of argument serialization and buffer management.

Table I
Average and standard deviation of network latency and bandwidth for local and wide area networks including Amazon EC2.

| | LAN | | Campus network | | Amazon EC2 | |
|---|---|---|---|---|---|---|
| Latency (*ms*) | Avg. 10.833 | Stdev. 2.684 | Avg. 15.465 | Stdev. 4.189 | Avg. 74.036 | Stdev. 17.737 |
| Bandwidth (*MB*) | Avg. 6.523 | Stdev. 0.177 | Avg. 2.461 | Stdev. 0.238 | Avg. 0.178 | Stdev. 0.023 |

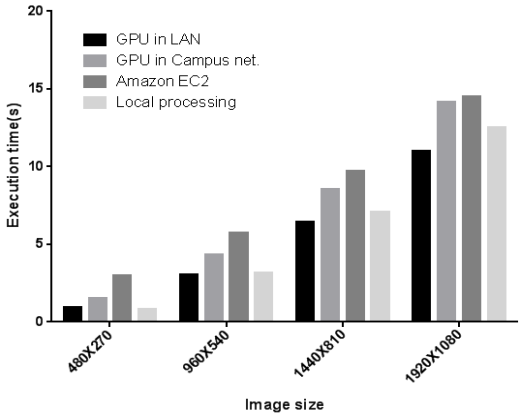
### B. Offloading Performance

In this subsection, we demonstrate the need for support from adaptive runtime schedulers by conducting an experiment in which we deploy our offloading framework subject to various network configurations and collect measurements to show the performance disparity between different network configurations.

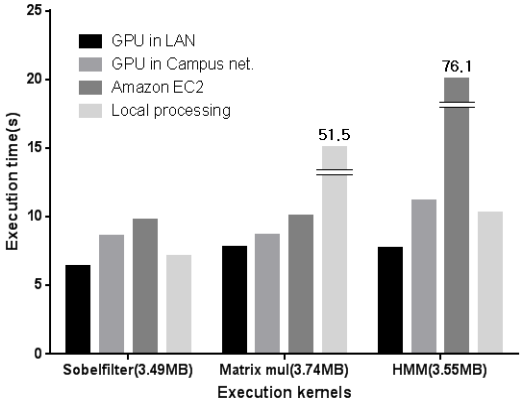
In the experiments, we utilized an Android tabletPC equipped with 1GHz dual-core processor and 1GB RAM as a mobile client. In order to observe the impact of different network conditions on the offloading performance, we deployed a remote server equipped with GeForce GT 640 graphics card into three different network configurations: local area network, campus network, and Amazon EC2 instance. In the local area network, we connected the mobile client and the remote server through a wireless router supporting 802.11 b/g/n network standard. The campus network is used to represent a wide area network in which the mobile client and the remote server are involved in different networks: the mobile client connects to the campus wireless router and the server connects to the laboratory internal router. They communicate each other through multiple routers in the campus. We used an Amazon EC2 GPU cluster as another option for the remote server located in a wide area network, but for more restricted network condition than the campus network. Table I summarizes the average and standard deviation of latency and bandwidth of the network configurations that we setup for the experiments.

The benchmarks used in the experiment are Sobelfilter, floating-point matrix multiplication, Hidden Markov Model, and *N*-body physics provided by AMD APP SDK [14] and Nvidia [15] sample code. These execution kernels are used by a variety of applications in areas such as image processing, physics simulation, and mathematical modeling.

Figure 2 shows the offloading performance in terms of




(a) Performance for Sobelfilter with different network configurations




(b) Performance differences between different workloads

Figure 2. Comparison of total execution time for four OpenCL workloads with various servers and network setup

the execution time compared to the case of local processing according to the data size and network configurations for four OpenCL execution kernels. As shown in Figure 2(a), for Sobelfilter, we observed that different network conditions result in significantly different offloading performance. Particularly, offloading to the remote server located in a local area network has better performance than local processing. In contrast, offloading to the remote servers located in the campus network and Amazon EC2 instance, where we have more restricted network conditions than a local area network, takes longer time than local processing. Figure 2(b) shows the performance difference among various execution workloads due to different computational requirements of workloads even though they process or offload the similar size of data ranging from 3.49MB to 3.74MB. For Sobelfilter, offloading to the GPU server located in LAN is only more beneficial than local processing. On the other hand, offloading floating-point matrix multiplication has always better performance than local processing in our setup due to heavier computational requirement of floating-point

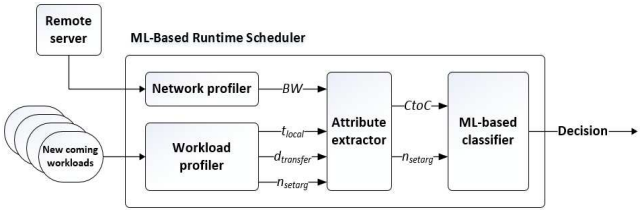



Figure 3. The structure of Machine Learning-based runtime scheduler

matrix multiplication. In fact, the computation complexity for floating-point matrix multiplication is $O(n^3)$ while that for Sobelfilter is $O(n^2)$.

It is also worth noting that, for Hidden Markov Model, offloading to Amazon EC2 instance shows the worst performance among other cases. This is because that Hidden Markov Model requires extra communications between the mobile client and the remote server to setup additional arguments for workload execution. Packets are exchanged at higher latencies in the Amazon EC2 setup compared with a local area network, which causes performance degradation since our offloading framework requires that each RPC call is acknowledged with a response from the remote server. Consequently, offloading to Amazon EC2 GPU instance, which has the highest latency among our experimental setups, takes the longest time. These results show that there is variation in offloading performance between different network conditions and execution workloads. Accordingly, proper scheduling can have a significant impact on the offloading performance, and remote offloading framework requires the support from the runtime scheduler.

## IV. Machine learning-based runtime scheduler for mobile offloading framework

In order to apply machine learning techniques to any decision-making problems, it is first required to select a subset of relevant attributes. These need to comprehensively represent a set of problem instances in terms of internal and external conditions which have an effect on making a decision. In this section, we describe the attributes of machine learning techniques considered in this paper, and how the proposed scheduler can extract these attributes. Then, using this subset of attributes, we investigate the scheduling accuracy of machine learning techniques using a dataset collected from experimental data using benchmark executions. By taking the text-based dataset as an input, we can train the classifier of various machine learning algorithms and examine the accuracy of the trained classifiers. Figure 3 illustrates the structure of our machine learning-based runtime scheduler and how it generates and uses the subset of attributes to make a decision for remote offloading framework.

### A. Selection of Machine Learning Attributes

Since offloading performance can vary as a function of network conditions, the size of data to be processed, and computational requirements, the scheduler has to take these factors into account to make an accurate decision on offloading or local processing. We focus on four features to establish the subset of attributes which is the representation of the scheduling problem for the remote offloading framework: (1)computation amount of the workload, (2)size of data, (3)network bandwidth, and (4)additional communication between the mobile client and the remote server to setup extra arguments.

**Local execution time($t_{local\_execution}$):** We regard the time for a workload to be executed in the mobile client locally as the computation amount. There are a variety of methods to measure the computation amount of the execution, such as counting the number of assembly instructions or loop iterations, some of which require additional assistance from the special hardware or compiler. Instead, in the proposed approach, runtime measurements are taken by the offloading framework as it executes the workload with the given data locally, and the scheduler profiles the execution time for the workload.

**Size of data to be transferred($d_{transfer}$):** In addition to the computation cost of a workload depending on the size of the data, the data size also affects the communication cost to transfer the data from the mobile client to the remote server. In our OpenCL-based remote offloading framework, the APIs for buffer management such as *clEnqueueWriteBuffer* and *clEnqueueReadBuffer* are used to profile the size of data to be transferred.

**Network bandwidth($BW$):** We integrate the network bandwidth measurement into the offloading framework so that it is able to measure network bandwidth between the mobile client and the remote server during runtime. In our implementation, network bandwidth is simply measured by the size of probing packets divided by the elapsed time to send those packets [16].

**Number of the invocations for argument setup($n_{argset}$):** We count the number of the invocations of the specific OpenCL API called *clSetkernelArgs*, which causes additional communication overhead between the client and the server to setup the extra arguments for kernel executions in addition to the primary data setup. The reason why we distinguish communications between main data transfer and additional arguments setup is that, though the latter incurs minor amount of data, it can cause significant communication costs due to protocol round-trip messages between the client and the server.

Note that, rather than considering the local execution time, the size of data transfer, and network bandwidth as individual attributes separately, we use *Computation-to-Communication ratio* in which three features are merged into one attribute as Equation 1.

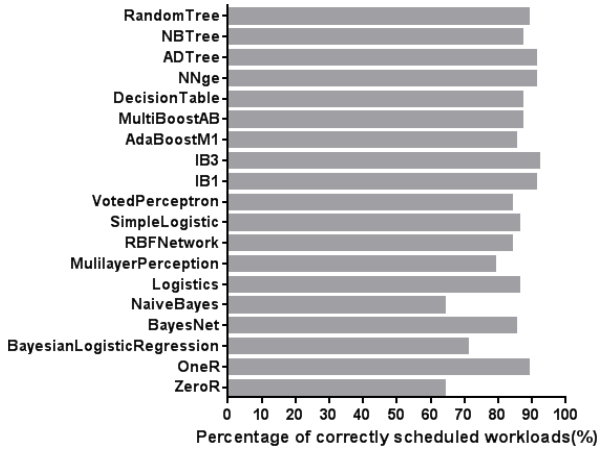



Figure 4. Offloading scheduling accuracy of various machine learning algorithms. The scheduling accuracy from the test dataset(30% of 640 dataset instances).

$$\begin{aligned} CtoC &= t_{local\_execution} \,/\, t_{data\_transfer} \\ &= t_{local\_execution} \,/\, (d_{transfer} \,/\, BW) \end{aligned} \quad (1)$$

where $t_{data\_transfer}$ is the time for data transfer. Thus, in our work, computation-to-communication ratio is a composite measurement which combines three dynamic features into one parameter. As a result, the proposed machine learning-based classifier accepts two attributes: computation-to-communication ratio(*CtoC*) and the number of invocations of *clSetKernelArgs*($n_{argset}$) to make a decision on scheduling a new workload (i.e. local processing or remote offloading).

### B. Scheduling Accuracy of Machine Learning Techniques

In this subsection, we investigate the scheduling accuracy of the machine learning-based scheduler for remote offloading framework. First of all, by running the OpenCL-based offloading framework into the experimental setup with various network configurations, execution kernels, and data sizes, we gathered a total of 640 data instances to train and test the classifiers of various machine learning algorithms. Each data instance means one execution of the offloading framework, and consists of $d_{transfer}$, $BW$, and $n_{argset}$. Next, we aggregate collected data instances to create the training and test datasets, each consisting of a 3-tuple, {*CtoC*, $n_{argset}$ ; *label*}, with two attributes explained in the previous section and the label which is a decision on *offload* or *local*. Then we labeled each data instance by comparing offloading performance to the local execution in terms of the execution time. For example, if offloading Sobelfilter with a 1920×1080 image into a machine with a GPU in LAN takes a shorter time than local processing, the instance is labeled as *offload*. In our collected dataset, 65% of instances are labeled as *offload*. Note that it is possible

to use another performance metric: mobile device's energy consumption, such that each instance can be labeled based on energy consumption between remote offloading and local processing. Lastly, we separated the collected dataset with 70% of them for the training dataset and the rest for the test dataset, so they have the identical distribution for instance properties.

Using the training dataset, we trained the classifiers of various categories of machine learning algorithms such as Decision Tree, Bayesian Networks, Instance-Based Learning, and Perceptron-Based Learning. In the training phase, Weka takes the text-based training dataset as the input and automatically generates the classifier associated with each machine learning algorithm. Once each classifier is trained, we tested the accuracy of the trained classifier with the test dataset by observing whether the trained classifier labels each instance in the test dataset correctly or not.

Figure 4 shows the scheduling accuracy of various machine learning algorithms. In this evaluation, the scheduling accuracy is calculated through the number of the correct decisions made by the classifier out of the test dataset. We observed that two Instance-Based Learning classifiers performed the most accurate prediction, showing greater than 90% of the scheduling accuracy. The basis for the classification of Instance-Based Learning is the instances database, where previously seen instances are stored. Instead of building the explicit classifier as other machine learning algorithms, Instance-Based Learning compares a new problem instance with the stored instances in the database to select $k$ most similar instances from the database and votes the majority of the selected instances to predict the label of the new problem instance. $k$ set to 1 (IB1) and 3 (IB3) as higher values showed the same performance. The classification of probabilistic machine learning techniques such as Bayesian Networks is based on the statistics of attributes of the previous instances such as the mean and the variance values. Thus it is possible that probability-based machine learning algorithms overlook the edge of previously seen instances, which causes the performance degradation for the prediction problem. In fact, Naive Bayes has the worst performance among machine learning algorithms used for the evaluation, showing 64.4% scheduling accuracy.

## V. Performance evaluation for offline scheduler

In this section, using the classifiers from various machine learning algorithms, we implement an offline offloading scheduler and evaluate the performance and penalty of the offline runtime schedulers.

### A. Experimental Setup

Based on scheduling performance and algorithm complexity, we selected three machine learning algorithms: RandomTree, Instance-Based Learning and Rule-Based Learning, and built them onto our remote offloading framework for the offline runtime scheduler. For RandomTree, we used the classifier that Weka generated using the training dataset described in Section IV.B. Total depth of the tree is 101 and the scheduling accuracy simulated through Weka is 89.5%. Though we do not need any classifier for Instance-Based Learning algorithm, it is required to define the similarity between a new problem instance and previously stored instances in the database. To do this, we used Euclidean distance, which is common to measure the similarity for Instance-Based Learning [17]. The closer the distance between instances is, the more similar they are. We stored the training dataset with 448 instances to build the database for the Instance-Based Learning algorithm. For simplicity, we use $k=1$ for Instance-Based Learning algorithm. For Rule-Based Learning, we establish a simple rule based on computation-to-communication ratio threshold, in which the scheduler decides to offload the mobile computation only if computation-to-communication ratio is higher than the threshold. Based on our observation, it is most likely that the benefits from offloading are more promising when computation-to-communication ratio is higher than 1.5. For that reason, we setup the threshold with 1.5 and 3.

Also, we emulate various network configurations in which the client and the server connect directly through a wireless router, but have 9 different network bandwidths ranging from 6.5MB/s to 0.3MB/s controlled by Traffic Control (TC) [18]. TC is a network tool which provides functionalities to control network traffic by prioritizing network resources and using concepts of traffic classification, queue disciplines and quality of service (QoS). While setting different network bandwidths, we ran our offloading framework with four benchmark kernels 720 times (9 network bandwidths $\times$ 4 kernels $\times$ 4 data sizes $\times$ 5 repeats for average) per each offline scheduler.

### B. Performance Comparison

Figure 5 shows the scheduling accuracy for various machine learning algorithms with four benchmark kernels. Similarly as the result shown in Figure 4, we observed that Instance-Based Learning has the most accurate scheduling performance among various schedulers showing 92% of the scheduling accuracy. Even though in matrix multiplication and Hidden Markov Model, other machine learning algorithms have better performance than Instance- Based Learning, Instance-Based Learning shows the best performance on average. It is also observed that, even though the schedulers based on Rule-Based Learning algorithm consider only one attribute, computation-to-communication ratio, they have better performance than RandomTree. An interpretation is that the computation-to-communication ratio is a more dominant attribute than the number of argument setup, $n_{argset}$. Interestingly, for $N$-body physics, all machine learning algorithms show the perfect scheduling accuracy. It

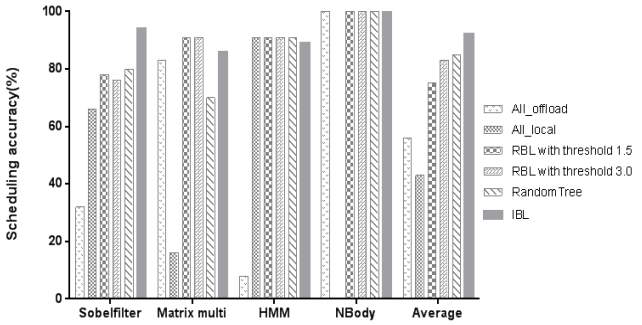



Figure 5. Scheduling accuracy of the offline schedulers using various machine learning algorithms. For performance comparison, we also added two simple scheduling policies, all_offload and all_local which are not machine learning algorithms.

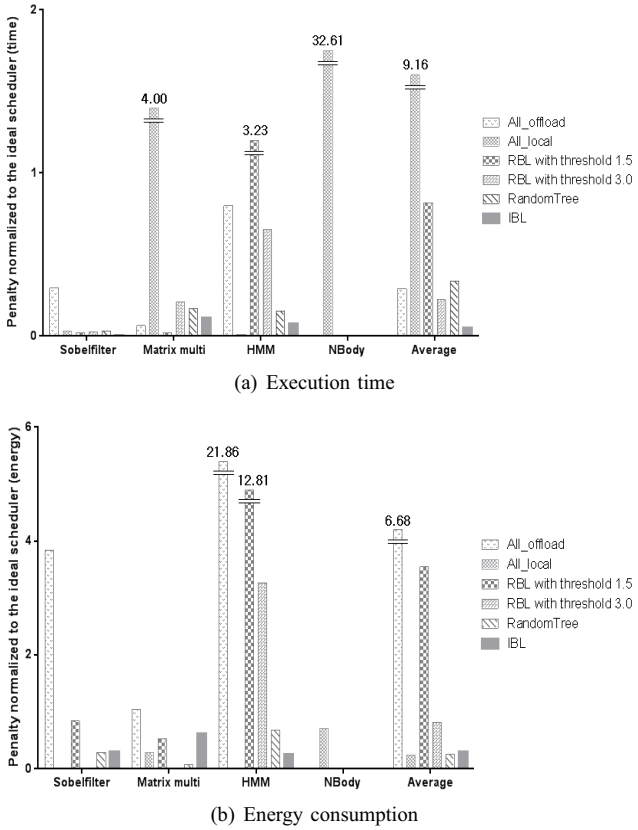
is because that computation-to-communication ratio for *N*-body physics is extremely high in our experimental setup so that it is easy for the scheduler to differentiate the conditions where offloading or the local execution for *N*-body physics is more beneficial than the other. In fact, offloading *N*-body physics always had better performance that the local execution in our setup.

Figure 6(a) and (b) present the penalty for various schedulers normalized to the case of the oracle scheduler which always makes the right decision to offload or run locally as Equation 2.

$$penalty_{normalized} = (execution\_time_{ML} - execution\_time_{oracle}) / execution\_time_{oracle} \quad (2)$$

where $execution\_time_{ML}$ and $execution\_time_{oracle}$ are the processing times of the workload scheduled by the machine learning-based scheduler and the oracle scheduler, respectively. For the penalty in terms of energy consumption, $execution\_time_{ML}$ and $execution\_time_{oracle}$ should be replaced with the mobile device's energy consumption to execute or offload the workload scheduled by each scheduler. In our evaluation, the penalty implies the extra costs that the mobile device or user has to pay additionally over the oracle scheduler when the machine learning-based scheduler makes a wrong decision. To profile energy consumption of the mobile device, we used PowerTutor [19] which is an application for the variants of Android devices that displays the power consumed by major components such as CPU, network interface, LCD display, and GPS receiver.

As you can see, the Instance-Based Learning scheduler has the smallest penalty in terms of the execution time because it has the highest scheduling accuracy. For energy consumption, moreover, the Instance-Based Learning scheduler has a fairly small penalty compared with Rule-Based Learning scheduler. Note that, for Sobelfilter, the penalty in terms of both the execution time and energy consumption




(a) Execution time

(b) Energy consumption

Figure 6. Penalty normalized to the oracle scheduler in terms of the execution time and energy consumption

is lower than other execution kernels, because the gap of the execution time and energy consumption for Sobelfilter between offloading and the local execution is relatively small. Therefore, the penalty for Sobelfilter is less significant than the cases of other execution kernels when the scheduler makes a wrong decision on offloading or the local execution.

## VI. Online runtime scheduler

In the previous section, we demonstrated the offline runtime offloading scheduler based on various machine learning algorithms by illustrating the scheduling accuracy and the penalty with regard to the execution time and energy consumption of the mobile platforms. In this section, we explore the potential possibility and benefits of an online runtime scheduler for mobile offloading framework in which the online scheduler can be trained through the previous experiences automatically and adapt to the dynamic situation.

### A. Implementation of the Online Offloading Scheduler

We first implemented the prototype of the online runtime scheduler based on the Instance-Based Learning algorithm for the mobile offloading framework. The reason why we

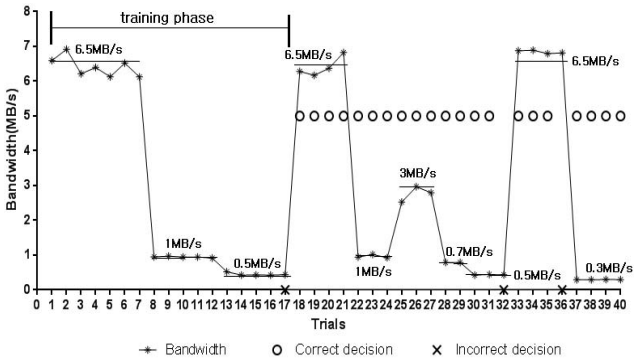



Figure 7. The ability to adapt dynamic network conditions for the online offloading scheduler

chose the Instance-Based Learning algorithm for the online runtime scheduler is due to the simplicity of the algorithm and the ability to quickly apply newly seen data to its future decisions. Usually, other machine learning algorithms such as neural networks or linear regression have its own the classification model and it is required to be completely modified when a new instance data is added to the training dataset. However, Instance-Based Learning simply stores the new instance to the training dataset, and the new instance is used to predict a next coming problem instance along with previous stored instances. In addition to its simplicity, in our evaluation for the offline offloading scheduler, Instance-Based Learning showed the best performance among various machine learning algorithms we used for the evaluation.

The following is the scheduling process of our prototype of the online scheduler. Once the application starts, the online scheduler executes the application locally at once to figure out the information which is required for profiling the workload such as the local execution time, the size of data, or the number of the invocations for argument setup. Then, the scheduler enters the training phase by unconditionally offloading the execution to the remote server $N$ times, and each case is labeled according to the performance comparison between offloading and the local execution. The labeled instance is stored into the training database. For our prototype, we set $N$ with 16. After the training phase, the scheduler starts the scheduling process by measuring the Euclidean distance between a new scheduling problem and the instances stored in the training database. When offloading is scheduled, the scheduler offloads the workload and measures the execution time for offloading. If offloading takes shorter than the local execution, then that instance is added to the training database as *offload*. On the other hand, it is stored as *local*.

To update the training database, the scheduler keeps adding the new instance to the database without removing any previous stored instance until the database is full. Only if the database is full, the oldest instance will be replaced with the new one. For our implementation, we try to find the optimal number of instances for the database which covers as many cases as possible, but takes reasonably small memory space (less than 1MB) and time (less than 0.01sec) to schedule a new problem. We chose 5,000 instances database which requires only 0.1MB of memory. This memory usage occupies only less than 0.0007% of typical memory sizes of contemporary mobile devices (e.g 16GB or 32GB). Also, even though it takes 5∼6msec to measure the Euclidean distance of the new scheduling problem with 5,000 instances, we believe that instance generalization or clustering techniques for the database such as [20], [21] can help the scheduler significantly reduce the measurement time.

### B. Evaluation for the Online Scheduler

In order to evaluate the prototype of our online scheduler, we conducted an experiment in which we change network conditions and observe how well the scheduler learns and adapts to dynamic network conditions. In this experiment, the client and the remote server are directly connected through a wireless router and we controlled the network bandwidth between them using TC. Also, we used Sobelfilter for the offloaded execution kernel. Figure 7 shows the ability to adapt the online scheduler to dynamic network conditions.

During the training phase, we setup different network conditions where the scheduler is trained with three network bandwidths: 6.5MB/s, 1MB/s, and 0.5MB/s. After the training phase, the scheduler makes a decision on offloading or the local execution in six different network bandwidths as shown in Figure 7. As you can observe, the scheduler makes the correct decisions in dynamic network conditions except for 17th, 32nd, and 36th trial. These incorrect decisions are because network bandwidth is changed after the scheduler makes a decision. As a result, the actual cost for data transfer is different with what the scheduler predicts. Furthermore, even at the unseen conditions in the training phase such as 3MB/s or 0.3MB/s, the scheduler works correctly by making right decisions. Consequently, we observed the possibility and the potential benefits of machine learning-based online offloading scheduler in this experiment.

## VII. Conclusion and Future work

In this paper, we proposed machine learning-based runtime scheduler for mobile offloading framework. Before addressing the scheduling problem of the mobile offloading framework, we showed the performance variance between different network conditions and workload characteristics by deploying the OpenCL-based offloading framework into a local and wide area network. By running various machine learning algorithms, we also showed the feasibility of adopting machine learning techniques into the scheduler problem for mobile offloading framework. In addition, we implemented an offline offloading scheduler using the classifiers

of RandomTree, Instance-Based Learning, and Rule-Based Learning. In the evaluation, the scheduler based on Instance-Based Learning performed 7% better than RandomTree and 3% better than Rule-Based Learning. Finally, using Instance-Based Learning, we demonstrated the potential benefits and the ability of the online offloading scheduler to adapt into dynamic network conditions.

As the future work, we can extend the online offloading scheduler by considering the scenario where multiple remote servers are available. By integrating the functionality to enumerate computing capabilities and network conditions for multiple servers, the scheduler can make a decision on not only offloading or the local execution, but also which remote server is the best to offload the mobile executions.

## ACKNOWLEDGMENT

This material is based upon work supported in part by the National Science Foundation under Grant No. 0910812, 0758596, 0855031, and 1265341. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

## REFERENCES

[1] T. Soyata, H. Ba, W. Heinzelman, M. Kwon, and J. Shi, "Accelerating mobile cloud computing: A survey," in *Communication Infrastructures for Cloud Computing*. IGI Global, 2013.

[2] H. Eom, P. S. Juste, R. Figueiredo, O. Tickoo, R. Illikkal, and R. Iyer, "Snarf: A social networking-inspired accelerator remoting framework." in *In proceeding of Workshop on Mobile Cloud Computing(MCC)*. ACM, 2012, pp. 29–34.

[3] E. Cuervo, A. Balasubramanian, D. ki Cho, A. Wolman, S. Saroiu, R. Ch, and P. Bahl, "Maui: making smartphones last longer with code offload." in *In proceeding of International Conference on Mobile Systems, Applications and Services(MobiSys)*. ACM, 2010, pp. 49–62.

[4] R. Kemp, N. Palmer, T. Kielmann, and H. E. Bal, "Cuckoo: A computation offloading framework for smartphones." in *In proceeding of International Conference on Mobile Computing, Applications and Services(MobiCASE)*, vol. 76. Springer, 2010, pp. 59–79.

[5] "The open standard for parallel programming of heterogeneous systems." [Online]. Available: http://www.khronos.org/opencl/

[6] "Weka: Data mining software in java." [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka/

[7] X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, and D. Milojicic, "Adaptive offloading inference for delivering applications in pervasive computing environments." in *In proceeding of International Conference on Pervasive Computing and Communications(PerCom)*, 2003, pp. 107–114.

[8] S. Imai and C. A. Varela, "Light-weight adaptive task offloading from smartphone to nearby computational resources." in *In proceeding of Symposium on Research in Applied Computation(RACS)*. ACM, 2011, pp. 146–152.

[9] D. Kovachev and P. Klamma, "Framework for computation offloading in mobile cloud computing." *International Journal of Interactive Multimedia and Artificial Intelligence*, vol. 1, no. 7, pp. 6–15, 2012.

[10] W. Alsalih, S. Akl, and H. Hassanein, "Energy-aware task scheduling: Towards enabling mobile computing over manets." in *In proceeding of IEEE International Parallel and Distributed Processing Symposium(IPDPS)*. IEEE, 2005, pp. 242.1–.

[11] Z. Wang and M. F. O'Boyle, "Mapping parallelism to multicores: A machine learning based approach." in *In proceeding of SIGPLAN Symposium on Principles and Practice of Parallel Programming(PPoPP)*. ACM, 2009, pp. 75–84.

[12] J. L. Berral, I. nigo Goiri, R. Rou, F. Julià, J. Guitart, R. Gavaldà, and J. Torres, "Towards energy-aware scheduling in data centers using machine learning." in *In proceeding of International Conference on Energy-Efficient Computing and Networking(e-Energy)*. ACM, 2010, pp. 215–224.

[13] J. Li, X. Ma, K. Singh, M. Schulz, B. R. de Supinski, and S. A. McKee, "Machine learning based online performance prediction for runtime parallelization and task scheduling." in *In proceeding of International Symposium on Performance Analysis of Systems and Software(ISPASS)*. IEEE, 2009, pp. 89–100.

[14] "Accelerated parallel processing(app) sdk with opencl 1.2 support." [Online]. Available: http://developer.amd.com/tools/heterogeneous-computing/amd-accelerated-parallel-processing-app-sdk/

[15] "Nvidia opencl sdk code samples." [Online]. Available: http://developer.nvidia.com/opencl

[16] C. Kozierok, *The TCP/IP Guide: A Comprehensive, Illustrated Internet Protocols Reference*. No Starch Press, 2005.

[17] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-based learning algorithms," *Journal of Machine Learning*, vol. 6, no. 1, pp. 37–66, 1991.

[18] W. Almesberger, "Linux network traffic control – implementation overview." 1999.

[19] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones." in *In proceeding of International Conference on Hardware/Software Codesign and System Synthesis(CODES+ISSS)*. ACM, 2010, pp. 105–114.

[20] P. Domingos, "Rule induction and instance-based learning a unified approach," in *In proceeding of International Joint Conference on Artificial Intelligence(IJCAI)*. Morgan Kaufmann Publishers Inc., 1995, pp. 1226–1232.

[21] C. L. Chang, "Finding prototypes for nearest neighbor classifiers," *IEEE Transaction on Computers*, vol. 23, no. 11, pp. 1179–1184, 1974.