ENDA: Embracing Network Inconsistency for Dynamic Application Offloading in Mobile Cloud Computing

Jiwei Li Kai Bu Xuan Liu Bin Xiao

Department of Computing
The Hong Kong Polytechnic University
{csjili, cskbu, csxuanliu, csbxiao}@comp.polyu.edu.hk

ABSTRACT

Mobile Cloud Computing (MCC) enables smartphones to offload compute-intensive codes and data to clouds or cloudlets for energy conservation. Thus, MCC liberates smartphones from battery shortage and embraces more versatile mobile applications. Most pioneering MCC research work requires a consistent network performance for offloading. However, such consistency is challenged by frequent mobile user movements and unstable network quality, thereby resulting in a suboptimal offloading decision. To embrace network inconsistency, we propose ENDA, a three-tier architecture that leverages user track prediction, realtime network performance and server loads to optimize offloading decisions. On cloud tier, we first design a greedy searching algorithm to predict user track using historical user traces stored in database servers. We then design a cloud-enabled Wi-Fi access point (AP) selection scheme to find the most energy efficient AP for smartphone offloading. We evaluate the performance of ENDA through simulations under a real-world scenario. The results demonstrate that ENDA can generate offloading decisions with optimized energy efficiency, desirable response time, and potential adaptability to a variety of scenarios. ENDA outperforms existing offloading techniques that do not consider user mobility and server workload balance management.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Distributed networks

Keywords

Mobile Cloud Computing, smartphones, offloading, user track prediction, cloudlet

1. INTRODUCTION

We are stepping into an era of advanced mobile computing that the widespread use of smartphones has moved people to a further stage of digital life. Unlike feature phones or PDAs, smartphones offer complete phone features and desktop-class application experience while maintaining good portability. Nowadays, people rely on their smartphones to do many things that would be hard to imagine several years ago, such as GPS-based navigation, social network interaction, on-line video streaming, etc. It is predicted that 1.2 billion mobile devices including smartphones and tablets will be sold in 2013 [4], which will open myriads of opportunities for manufacturers, application developers and mobile operating system designers.

Though capable of running advanced PC-like applications, smartphones suffer severely from limited battery life due to their inherent small size and slowly-evolving lithium battery technology, which greatly encumbers innovative ideas and imagination of application developers. Fortunately, rapid development in wireless network technology and cloud computing [6, 7, 9] allows smartphones to effortlessly access nearby computing facilities (i.e. cloudlet) [8] or remote server clusters (i.e. data centers) [1]. Thus, one intuitive solution towards battery shortage is to offload compute-intensive codes of an application to third-party computing facilities [2, 3, 5]. Normally, good network performance allows mobile applications to benefit from offloading in terms of energy consumption and response time.

Existing offloading techniques (such as MAUI [3] and CloneCloud [2]) assume that network performance, especially Wi-Fi, remains consistent throughout the offloading process. However, keen observations prove the opposite in a real-world environment. Consider a scenario where a user holding a Wi-Fi connected smartphone walks from one room to another wall-separated room. As the user walks into the second room, the Wi-Fi signal from the first room is weakened by the blocking walls, thereby leading to higher network transmission overheads and even network disconnection. The optimal offloading decision made in the first room becomes suboptimal. In such case, an application would have conserved more energy if executed on the smartphone, compared to being offloaded to other computing facilities. Inconsistency in network performance undermines the positive effects of offloading techniques, which calls for more efficient mechanisms to make dynamic and accurate offloading decisions. The problem becomes even more challenging when network performance is affected not only by user mobility, but also by server overload to receive dynamically changing number of requests.

In this paper, we attempt to solve the problem of how to make the most cost-effective offloading decisions for smart-

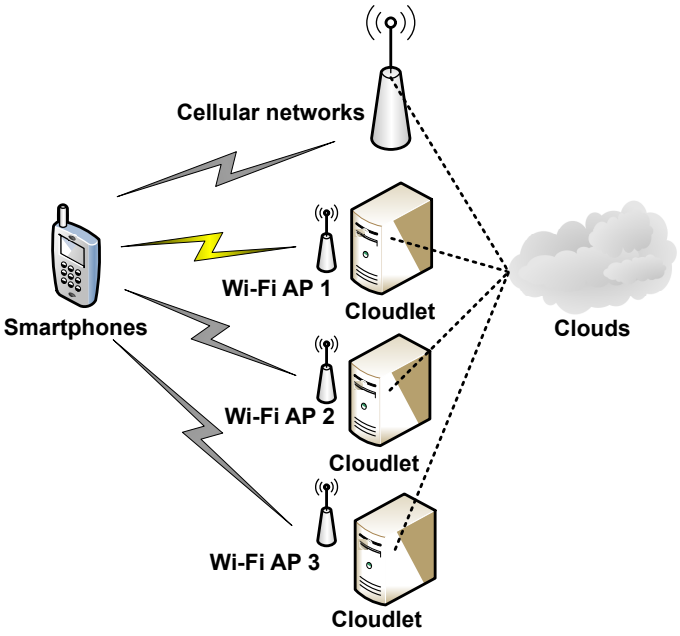

Figure 1: Smartphones can offload applications to cloudlets through Wi-Fi, to clouds through cellular networks, or execute them locally. The offloading decision usually minimizes energy consumption on smartphones while satisfying certain conditions.

phone applications, considering user mobility, realtime network performance and server loads. Specifically, smartphones need to know which network to connect to and which computing facility to offload to (see Fig. 1), with objectives of minimizing overall execution overheads on smartphones while satisfying certain conditions (such as maximum response time).

To this end, we propose a three-tier architecture called ENDA, i.e. smartphones, cloudlets and clouds, where components on each tier interact with one another so as to optimize offloading decisions in constantly changing environment. ENDA has several advantages compared with previous approaches. First, it introduces user track prediction to dynamically estimate user moving direction and track based on current locations and historical user traces in database servers on clouds. This helps further lower the chances of network performance downgrading and network disconnection caused by frequent user movements. Second, ENDA considers server-side loads and network quality (mainly of cloudlets), and implements a simple management mechanism to balance workloads among cloudlets. Third, most operations in ENDA are transparent to smartphones, and smartphones only need to communicate very few data with servers on clouds. This can further reduce energy consumption on smartphones. We summarize our contributions as follows.

1. We propose ENDA, a novel three-tier architecture that involves smartphones, cloudlets and clouds into the process of making offloading decisions, which is different from existing approaches that only use smartphones. Such division of labor can further reduce energy consumption on smartphones. Most complex computations and continuous environment profilers are executed on clouds or cloudlets. Smartphones are only required to transmit very few data.

2. ENDA considers user mobility, realtime network performance and server loads. Therefore, ENDA can make more efficient offloading decisions than existing approaches that ignore the issue of network inconsistency. In addition, ENDA implements a fault-tolerant mechanism to handle inaccurate offloading decisions.

3. We design a greedy searching algorithm to predict user track based on user traces collected in database servers. We also design a Wi-Fi AP selection scheme that iteratively seeks the most energy efficient Wi-Fi access point for offloading.

4. Preliminary simulation results prove that ENDA can make better offloading decisions in terms of energy efficiency under different circumstances, compared with previous approaches that randomly select network service providers.

Next, we introduce the background of application offloading methods and motivation of our work in Section 2. Section 3 first presents a high-level overview of ENDA system architecture, and then discusses three important components of ENDA in detail. In Section 4, we also explain the fault-tolerant mechanism adopted in ENDA. Evaluation and conclusion can be found in Section 5 and Section 6, respectively.

2. BACKGROUND AND MOTIVATION

Compute-intensive applications, if executed locally on smartphones, usually demand a large quantity of CPU cycles, thereby draining batteries fairly fast. By virtue of ubiquitous access to servers through wireless networks, applications on smartphones can now be offloaded partially or completely to nearby cloudlets or distant clouds, thus saving energy while achieving desirable response time. Current offloading techniques usually employ static analyzer, dynamic profiler and optimization solver that are all implemented on smartphones to cooperatively make optimized offloading decision [2, 3, 5]. Static analyzer is used to mark which parts within application codes can be offloaded. Dynamic profiler monitors smartphone hardware usage, application information and network performance. The outputs of static analyzer and dynamic profiler serve as input for optimization solver that makes the final optimized offloading decisions.

Nevertheless, existing approaches implicitly assume that smartphones are connected to networks of constant latency and bandwidth. We argue that wireless network performance is inconsistent due to the inherent mobility of smartphone users. It is most likely that optimization solver would generate offloading decisions that should have been beneficial, but actually drain more batteries since user mobility causes network disconnection. Note that once disconnected from network, the offloaded application waits for reconnection, thus prolonging response time, or is re-executed, thus consuming more energy. We propose to utilize GPS, a feature commonly seen in smartphones, to collect user traces and store them in database servers on clouds. They will be compressed into useful information that can be used to predict user track. We believe that integrating user track

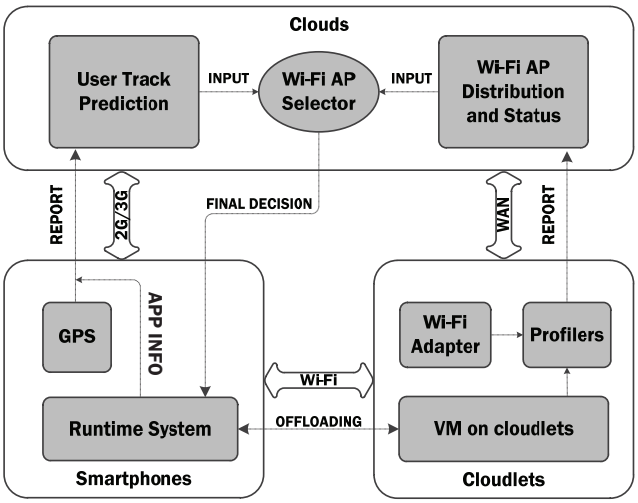

Figure 2: ENDA System Architecture

into the process of making offloading decision improves the efficiency of application offloading, as it will not choose networks that may disconnect smartphones.

3. ENDA ARCHITECTURE

ENDA is a three-tier architecture, i.e. smartphones, cloudlets and clouds, where components on each tier interact with one another, aiming at optimizing the offloading decisions for smartphone applications. The key idea is to select the most energy efficient network for application offloading based on user track prediction and other environment factors. Through integrating user mobility, network quality and server load into decision making, ENDA minimizes the unnecessary overheads of data re-transmission or program re-execution, which usually result from network performance downgrading or network disconnection.

In ENDA, smartphones are required only to communicate very few data with clouds, including application information and geographical locations, thus preserving more energy than previous approaches of placing the whole decision making on smartphones. In order to facilitate user track prediction, traces of smartphone users are usually collected and stored in database servers on clouds. We argue that as nowadays it is trivial to locate smartphones users through either GPS or cellular network, user trace collection is not of technical challenges, but more of moral and privacy issues. Users may be required to sign agreement to avoid such issue.

ENDA shifts the responsibility of profilers from smartphones to cloudlets, to save energy on smartphones and to monitor environment factors more accurately, including server loads and network performance. These cloudlets are presumably connected with public Wi-Fi stations, and the distribution map of Wi-FI APs is always known (e.g. the PCCW company provides thousands of Wi-Fi APs covering most regions in Hong Kong as shown in Fig. 3). We argue that it is possible for clouds to maintain the same distribution maps of cloudlets as of Wi-Fi. Thus, based on smartphone users' locations, clouds are able to query cloudlets for realtime environment factors through wired network.

The most complex computations in ENDA are executed on clouds, including user track prediction and Wi-Fi AP

Figure 3: Google Map for PCCW Wi-Fi Locations in Hong Kong

selection. To improve accuracy of user track prediction, ENDA uses database servers on clouds to collect user traces, and implements aggregation algorithms to compress them to useful routes. When a smartphone initiates an offloading request, ENDA attempts to find a route that matches most the smartphone's reported locations. We design a Wi-Fi AP selection scheme to filter out those networks that are either connected with overloaded cloudlets or will disconnect smartphones at some point. Then the selection scheme chooses one qualified network with lowest latency while achieving specified response time. Fig. 2 provides a high-level overview of the ENDA architecture.

In the next few sections, we give a more detailed description of important components in ENDA. First, we discuss cloudlets and how profilers monitor environment factors in Section 3.1. Followed is an introduction of algorithms used in user track prediction in Section 3.2. In Section 3.2, we present the component of Wi-Fi AP selector that implements our designed Wi-Fi AP selection scheme.

3.1 Cloudlets and Profilers

Offloading applications to nearby cloudlets through Wi-Fi is arguably more energy efficient than to distant clouds, since wireless local area network usually has lower latency that benefits latency-sensitive offloading more. Nevertheless, cloudlets are usually limited in computing capability and storage capacity. In order to guarantee quality of service, cloudlets must constrict the number of offloading requests and shall not accept new requests if recourse usage exceeds specified percentage. On the other hand, as more smartphones are offloading applications to a cloudlet, average bandwidth per user is bound to decrease, prolonging the time needed to transmit the same size of data (i.e. response time). Signal collision and network congestion also increase latency, which affects adversely the offloading overheads (as shown in Table 1 [3]).

Unit: mj	Wi-Fi(25ms)	Wi-Fi(50ms)	3G(220ms)
10kB	330	542	1573
100kB	576	979	2762

Table 1: Offloading overheads are mainly related to network latency and offload data size.

In ENDA, we implement profilers on cloudlets, which can monitor load factor, average bandwidth per user and network latency more accurately. Since an offloading request triggers a Virtual Machine (VM) to run on cloudlets, we refer to the ratio of the number of running VMs versus overall VM capacity as the load factor ρ of cloudlets. We refer to average bandwidth per user as B, and latency as L. When a query from clouds arrives, cloudlets usually respond with specific values in $<\rho, N, B, L>$, where N is the Wi-Fi access point name.

3.2 User Track Prediction

In ENDA, user track prediction is one important module implemented on clouds. It predicts the most probable user track based on reported user locations and collected user traces in database servers. These collected user traces are stored in the form of 2-D coordinate $<x, y>$, with tag information of time and date. Simple aggregation algorithms are implemented to process these traces so that a list of routes $List<R>$ is generated. Each route in $List<R>$ is comprised of a series of geographically continuous coordinates $List<x, y>$.

We cannot guarantee finding a route in $List<R>$ which matches exactly with reported user locations, because the GPS-based location measurement is of errors. Thus, we propose a greedy searching algorithm to predict user track. The goal is to find the route that is geographically nearest to the reported locations. We set a distance threshold α to filter out apparently irrelevant routes so as to improve the searching efficiency. Let $<x', y'>$ denote location coordinate reported by the user and $List<x, y>$ a list of location coordinates for each route $<R>$ stored in database servers. Then the route $<R>$ to be considered qualified must satisfy that there exists at least one coordinate $<x, y>$ such that $d = \sqrt{(x'-x)^2 + (y'-y)^2} \leq \alpha$. Alg. 1 presents a function called *FindRoutes* which inputs $List<R>$ and outputs a list of qualified routes $List<R'>$.

Algorithm 1: FindRoutes($List<R>$, $<x', y'>$, α)

Input: $List<R>$: a list of routes to be compared with, $<x', y'>$: reported location, α: distance threshold
Output: $List<R'>$: a list of routes found qualified
$List<R> \leftarrow null$;
for *all* $R \in List<R>$ **do**
  for *all coordinate* $<x, y>$ *in* R **do**
    `// Calculate distance between reported and stored location`
    $d = \sqrt{(x'-x)^2 + (y'-y)^2}$;
    if $d \leq \alpha$ **then**
      add this R to $List<R'>$;
      break;
    end
  end
end
Return $List<R'>$;

To improve prediction accuracy, the user is required to report not only current location but also previous continuous locations. We differentiate these locations by adding a subscript i indicating the number of locations reported. We refer to the subscript $i-1$ as the current report while 0 as the most previous. *FindRoutes* is iteratively invoked on each location coordinate in the list of reported locations denoted as $List<x', y'>$. The final output is the list of qualified routes. For each qualified route, its distances with each reported location are calculated and then aggregated. These aggregated distances are then compared, and the route with the smallest one is chosen as the ultimate output R_{min}. Alg. 2 formally presents the algorithm.

Algorithm 2: Finding the smallest aggregate distance

Input: $List<x', y'>$: a list of reported locations, $List<R>$: a list of all routes stored in the database, α: distance threshold
Output: R_{min}: the route with the smallest aggregate distance
$R_{min} \leftarrow null$;
$i \leftarrow List<x', y'>.length$;
$d_{min} \leftarrow i * \alpha$;
`// Find qualified routes`
while $i \geq 1$ **do**
  $List<R> \leftarrow$ FindRoutes($List<R>$, $<x'_{i-1}, y'_{i-1}>$, α);
  $i \leftarrow i - 1$;
end
`// Find route with smallest aggregate distance`
for *all* $R \in List<R>$ **do**
  Calculate the aggregate distance d between $List<x', y'>$ and R;
  if $d < d_{min}$ **then**
    $R_{min} \leftarrow R$;
    $d_{min} \leftarrow d$;
  end
end
Return R_{min};

3.3 Wi-Fi AP Selector

The goal of Wi-Fi AP selector is to find the most energy efficient offloading Wi-Fi AP and, if any, forward it back to the requesting smartphone. Fig. 4 illustrates an example of a smartphone user faced with multiple Wi-Fi APs. The design of Wi-Fi AP selector follows three fundamental principles: 1)achieving workload balance among cloudlets, 2)avoiding Wi-Fi disconnection due to user mobility, and 3)minimizing transmission overheads through Wi-Fi. To be specific, the selector declines cloudlets with load factor $\rho \geq 80\%$ (default) so that workloads can evenly be distributed among all participating cloud-lets. Then, the selector filters out Wi-Fi APs that cannot maintain connectivity with smartphones along the estimated route, in order to avoid Wi-Fi disconnection. Finally, the selector chooses the Wi-Fi AP with lowest latency while achieving desirable response time, so as to minimize the overheads of the latency-sensitive offloading. Note that *except application data size, latency is considered as one significant factor to measure offloading overheads* (see Table. 1).

The outputs of profilers on cloudlets and user track prediction, namely $List<\rho, N, B, L>$ and R_{min}, as well as application information reported by the user, combine to serve as the inputs for Wi-Fi AP selector. We describe how Wi-Fi AP selector works as follows.

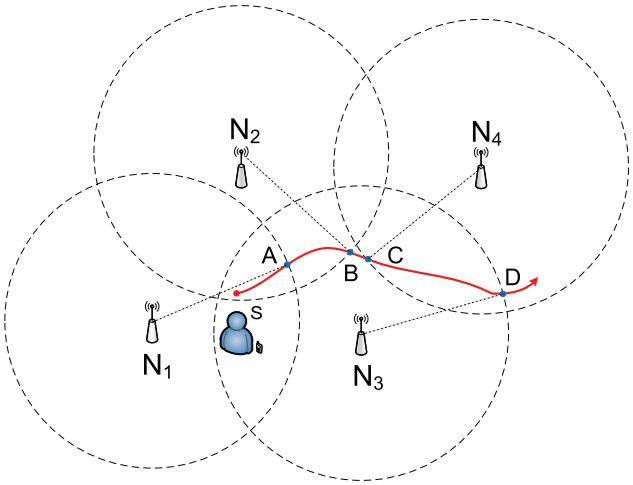

Figure 4: An example of a smartphone user faced with multiple choices of Wi-Fi APs. Note that we do not deem Wi-Fi AP N_4 qualified, although it covers the latter part of the user track.

1. The selector filters out Wi-Fi APs with $\rho \geq 80\%$ (default) in $List<\rho, N, B, L>$. The new list is denoted as $List'<\rho, N, B, L>$.
2. The selector calculates the overall time t_j needed to complete the whole offloading for each Wi-Fi AP N_j in $List'<\rho, N, B, L>$. Normally, $t_j = D/B_j$, where B_j is average bandwidth per user and D is data size to be offloaded.
3. For each Wi-Fi AP N_j in $List'<\rho, N, B, L>$, the selector continues calculating the distance d_{jk} between N_j's base station location and each coordinate $<x_k, y_k>$ in the estimated route R_{min}, until $d_{jk} \geq r_{max}$, where r_{max} is the radius of the maximum coverage circle of N_j to transmit data effectively. Then the selector calculates the route length l_j between the starting coordinate $<x_0, y_0>$ and the ending coordinate $<x_k, y_k>$, and the time $t'_j = l_j/s$ needed for the user to walk the length l_j with constant walking speed s.
4. The selector further filters out Wi-Fi APs with its overall time t_j greater than t'_j. The new list is denoted as $List''<\rho, N, B, L>$.
5. The selector chooses the Wi-Fi AP N_j with lowest latency in $List''<\rho, N, B, L>$.
6. If $t_j \leq t_{app}$, where t_{app} is the maximum response time for the application, then the selector will forward this Wi-Fi AP N_j back to the smartphone. If not, remove N_j out of $List''<\rho, N, B, L>$ and return to Step 5.

Normally, Wi-Fi AP selector can find at least one Wi-Fi AP that satisfies the conditions of keeping Wi-Fi always connected and response time shorter than specified. In the case of failing to find any qualified Wi-Fi AP, the selector would notify the requesting smartphone to either offload through cellular networks or execute applications locally. In our future work, we will introduce a delay offloading mechanism to improve the success rate of finding qualified Wi-Fi AP. It is possible that some Wi-Fi APs does not cover the starting part of the user track, but will do as the user moves along the track. Thus, under the delay offloading mechanism, these APs will be considered qualified and then compared with other qualified APs.

4. FAULT-TOLERANCE DESIGN

In ENDA, there exist a number of factors to mislead Wi-Fi AP selector, resulting in making suboptimal or even inaccurate offloading decisions. These factors can be mistakenly predicted user track, variable network latency and unstable bandwidth, etc. Because user track is predicted based on collected user traces, it is difficult to guarantee that users follow the predicted user track every time. During the offloading process, it is also difficult to guarantee that network performance remains unchanged. Hence, we propose a fault-tolerant mechanism to tackle these problems.

The fault-tolerant mechanism is designed mainly to handle mistakenly predicted user track and network disconnection. First, we propose a remedy that will initiate another offloading request to clouds, if the user's subsequent locations do not fit in the predicted track during specified tolerating time. However, if track mismatch is identified after tolerating time, no new offloading requests will be initiated. Second, we adopt similar methods for network disconnection as in previous offloading techniques, i.e. either to wait for re-connection with previous Wi-Fi AP, or to initiate a new request to clouds for offloading decisions.

5. EVALUATION

To evaluate ENDA, we simulate a scenario where a smartphone user walking in a constant speed would like to offload applications to cloudlets through Wi-Fi. We assume that the estimated user track, the distribution map of Wi-Fi APs and their bandwidth and latency are given in advance. The objective of this simulation is to compare the energy efficiency of offloading decisions made by ENDA and existing approaches. We wrote the simulation source codes in graphical user interface with over 300 lines of Java language.

In the simulated scenario, we set the number of Wi-Fi APs available as four, their maximum radius of effective coverage circle as uniformly 100 m. We also record the available bandwidth and latency of each Wi-Fi AP in Table 2. Meantime, we specify the estimated track and direction that the user would follow. Fig. 5 is a figure generated by our simulation program according to parameters above. Note that the track direction is from left to right.

	Wi-Fi A	Wi-Fi B	Wi-Fi C	Wi-Fi D
Bandwidth	1 Mbps	2 Mbps	1.5 Mbps	3 Mbps
Latency	50 ms	25 ms	75 ms	25 ms
Route covered	85 m	38 m	205 m	136 m

Table 2: Available bandwidth, latency and covered route length for each Wi-Fi AP

In our experiments, we vary the application data size from 1 Mb to 1000 Mb, to observe how offloaded data size would affect the final offloading decisions in ENDA. First, the algorithms implemented in Wi-Fi AP selector generates the results of the covered route length for each Wi-Fi AP, which are also recorded in Table 2. Obviously, the selector excludes

Figure 5: The simulated scenario where a smartphone user is faced with multiple Wi-Fi APs

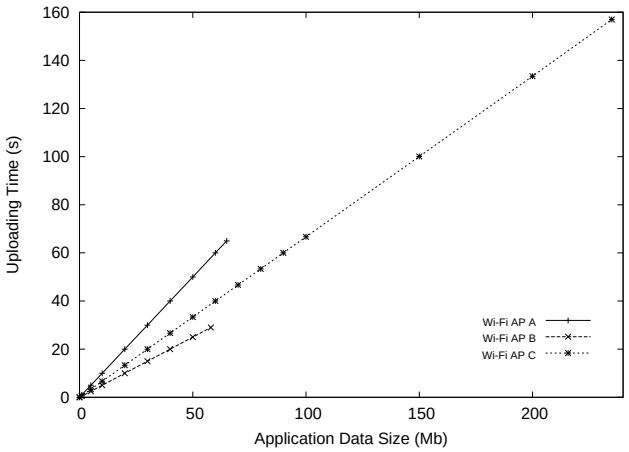

Figure 6: Offloading time of different application data sizes for each qualified Wifi AP

Wi-Fi AP D out of the list of qualified APs, since it cannot cover where the user starts. Assuming that users walk in an average speed of 1.3 m/s, the selector can easily calculate the maximum offloading times for each qualified Wi-Fi AP, which are respectively 65 s, 29 s, and 157 s. Fig. 6 illustrates how offload time changes as application data size varies for different Wi-Fi AP. Note that the end of each curve corresponds to the maximum offloading times of each Wi-Fi AP.

As offloading overheads are affected largely by network latency as shown in Table 1, we conclude the simulated results using latency as the only metric. For application data size less than 58 Mb, all Wi-Fi APs are qualified and Wi-Fi AP B is chosen due to its lowest latency 25 ms. When application data size falls into range between 58 Mb to 65 Mb, Wi-Fi AP A and C are qualified, and A is chosen because it has lower latency than C, although A has a lower bandwidth. If application data size exceeds 65 Mb but less than 235 Mb, Wi-Fi AP C is the only option left. Once application data size is greater than 235 Mb, the selector will immediately inform the requesting smartphone to either execute the application locally or wait a few seconds to request again.

Existing approaches select Wi-Fi APs randomly, which is different from ENDA. Without considering user mobility and network quality, they would choose each AP (i.e. A, B and C) uniformly at random regardless of different application data size. Under this scenario, as application data size varies, existing approaches would yield the most energy efficient decisions only with a probability of 1/3, while ENDA can generate optimal decisions that adapt to nearly all data sizes.

6. CONCLUSIONS

In this paper, we propose ENDA, a three-tier architecture that makes adaptable energy efficient offloading decisions in constantly changing environment, such as frequent user movements, varying server loads and network performance. ENDA places most operations of decision making on clouds and cloudlets, and only requires smartphones to communicate very few data with clouds. We present how ENDA predicts user track based on collected user traces, and how it selects the qualified Wi-Fi APs. ENDA has the potential to be practically applied in real mobile cloud systems. Our preliminary results demonstrate that ENDA outperforms existing approaches in terms of energy efficiency of offloading decisions. Our future work will focus on developing delay offloading mechanism and utilizing multiple Wi-Fi APs to accommodate applications with overlarge offload data size. We will also verify the effectiveness of our user track prediction algorithms and implement ENDA in a real-world environment.

7. REFERENCES

[1] P. Bahl, R. Han, L. Li, and M. Satyanarayanan. Advancing the state of mobile cloud computing. In *Proc. of ACM MCS*, 2012.

[2] B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: Elastic execution between mobile device and cloud. In *Proc. of ACM EuroSys*, 2011.

[3] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. MAUI: Making smartphones last longer with code offload. In *Proc. of ACM MobiSys*, 2010.

[4] Gartner. *http://www.gartner.com/*.

[5] M. Gordon, D. Jamshidi, S. Mahlke, Z. Mao, and X. Chen. Comet: Code offload by migrating execution transparently. In *Proc. of USENIX OSDI*, 2012.

[6] Y. Hua, B. Xiao, and X. Liu. Nest: Locality-aware approximate query service for cloud computing. In *Proc. of IEEE INFOCOM*, 2013.

[7] D. Huang. Mobile cloud computing. *IEEE COMSOC Multimedia Communications Technical Committee (MMTC) E-Letter*, 6(10):27–31, October 2011.

[8] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlet in mobile computing. *IEEE Pervasive Computing*, 8(4):14–23, October 2009.

[9] P. Shankar, B. Nath, L. Iftode, W. Huang, and P. Castro. Crowds replace experts: Building better location-based services using mobile social network interactions. In *Proc. of IEEE PerCom*, 2012.