From Mobiles to Clouds: Developing Energy-aware Offloading Strategies for Workflows

Bo Gao[1], Ligang He[1], Limin Liu[2], Kenli Li[3] and Stephen A. Jarvis[1]
1. Department of Computer Science, University of Warwick, Coventry, CV4 7AL, UK
2. Department of Optical and Electronic Engineering, Mechanical Engineering College, Shijiazhuang, China
3. School of Computer and Communication, Hunan University, Changsha, China
liganghe@dcs.warwick.ac.uk

***Abstract* - Cloud computing and mobile computing are two of the most influential technologies that look set to change the face of computing in the coming years. Combination of the two provides us with an unprecedented opportunity to provide highly portable and yet content-rich and computation-intensive services to the end user. In this paper we investigate the possibility of using code/task offload techniques between mobile and cloud in order to reduce the energy cost of workflows deployed on mobile devices. We first present a vision in which mobile devices are coordinated over a network, which is equipped with a layer of cloud-like infrastructures which we term *cloudlets*, whose computational resources can be leveraged by the mobile devices to host the execution of mission-critical mobile workflows in an energy-aware manner. We then build a model that encompasses various characteristics of the workflow's software and the network's hardware devices. With this model, we construct the objective functions that guide the offload decisions. We then present a heuristic algorithm that produces statistical and dynamic offload plans according to these objective functions and their variations both statically and dynamically. We conclude the paper with a series of simulation studies, the results of which give insight into the offload-ability of workflows of different characteristics. The results also illustrate how different hardware specifications can affect offload efficiency. These studies indicate that our offload algorithm can significantly improve the energy efficiency and execution speed of mobile workflows.**

Keywords-Workflow; code offload; energy; mobile computing;

I. INTRODUCTION

A mobile workflow, as presented in this paper, consists of a sequence of interactive tasks that are deployed over a network of distributed mobile devices. As suggested in [1], an organisation is able to rely on the computing and connectivity capabilities within the mobile devices as a substitute to a technology back end server infrastructure. In [2], scenarios are used to demonstrate how a mass of mobile devices, each used as a rich sensor, can be used to solve real-life problems that could not have been solved by traditional methods. With the ability to collect and process data anywhere and at anytime, applications deployed over a network of mobile devices provide the user with much more flexibility than the traditional desktop-based work environments.

Indeed, mobile devices are becoming the platform of choice for both enterprise and personal computing needs. It is predicted that by 2015, mobile application development projects will outnumber native PC projects by a ratio of 4:1 [3]. In recent years, the mobile platform's ability to enable ubiquitous access to services on the move has broadened the usability of many social and entertainment media and created great successes.

With the rapid development of the smartphone and tablet market comes a new generation of handheld devices equipped with powerful processing units and high quality display units that have not been seen before in the mobile world. With this improved hardware capability, sophisticated, intelligent and mission-critical processes will be adapted from desktops to the mobile platform, and we also expect to see novel applications utilising the unique features of the devices developed for the mobile world. However, in order to achieve that goal, there are several technical challenges, including solutions to increase mobile devices' battery life. In comparison to other components, the pace of advancement in improving the energy density of its battery has been slow [4]. Furthermore, gains made at a hardware level have often been taken up by extended software functionalities [5].

In this paper, we look at code/workload offload to reduce the energy cost on mobile devices in the execution of mobile workflows. This approach is based on well-known methods adapted from the desktop environment. However, early research has been constrained by the lack of fast and ubiquitously accessible offload platform and thus has focused on partitioning programs statically [6] [7] [8].

A new layer of network infrastructure called a *cloudlet* is the term used to capture the offload destination in this paper. The concept of a cloudlet was first introduced in [9] at the end of the last decade, and was subsequently discussed in [2] [10] and [11]. In [9], a cloudlet is described as a "data centre in a box" and is "self-managing, requiring little than power, Internet connectivity, and access control for setup." In Figure 1, we present an example in which a cloudlet is deployed next to a WiFi hotspot in a coffee shop that is accessible to the user of the second smartphone. In this example, we have a workflow that consists of four consecutive tasks deployed on four different devices. We assume that all tasks require the same amount of energy per second to run on their *host* devices and that the communications between each task are of the

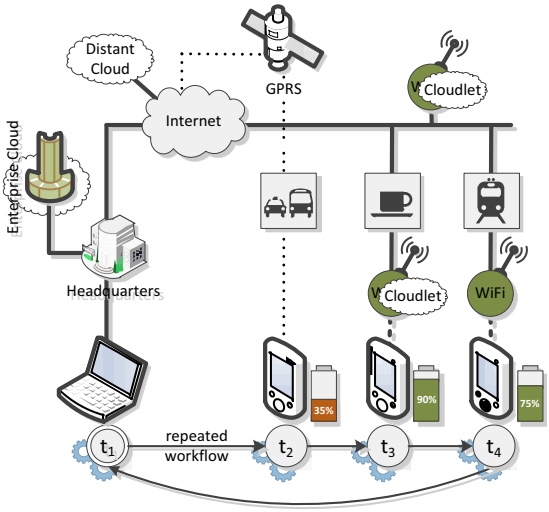


Figure 1: *Example showing cloudlet and faster network connections improve battery life on mobile devices.*

1550-5510/12 $26.00 © 2012 IEEE
DOI 10.1109/Grid.2012.20

same size. We also assume that no other application draws energy from these devices whilst the workflow is being executed. The workflow is run repeatedly, and we calculate the first phone's battery to go flat first because its user is sitting in traffic and can only communicates with the other phones over a 3G connection, and its offload activity (if any) also has to go through a 3G connection (3G is more expensive than WiFi [12] [13] [14]). The second phone communicates with others over the coffee shop's WiFi and is able to use its cloudlet to offload some of t_3's computation, and so its battery gets consumed the slowest. The user of the last handset has access to a WiFi hotspot whilst travelling on the train. However the train does not have a cloudlet deployed, so to offload t_4's computation it has to send the executables to a more distant cloudlet which takes longer to reach and thus consumes more energy. Note that in Figure 1, an enterprise cloud at the firm's headquarters and a distant cloud service on the Internet are also available to support offload. These nodes may have faster processing speeds than the cloudlets. Offload to these clouds could prove more beneficial if the network connection is of high speed.

In the remainder of this paper, we first discuss the related work and common approaches to implementing the mobile offload architecture; see section II. In III we present our offload model and discuss our algorithm with its features and variations. We conclude the paper with a parametric simulation study, see IV, in which we present the impact of different software and hardware metrics over the offload-ability and effectiveness of a mobile workflow.

II. Related Work

The idea of transferring computation to a nearby processing unit in order to improve mobile application's performance and reduce local energy cost has been researched along with the maturity of mobile technologies. Many ideas and techniques we use in this paper are inspired by this work.

Early research focuses on the partition schemes of an application. Aimed at energy management, a compile-time framework supporting remote task execution was first introduced in [8]. Based on the same approach, a more detailed cost graph was used in [7] with a parametric analysis on its effect at runtime presented in [15]. Another compiler-assisted approach was introduced in [16], which turns the focus to reducing the application's overall execution time. Spectra [17] adds application fidelity (a run-time QoS measurement) into the decision making process and uses it to leverage execution time and energy usage in its utility function. Spectra monitors the hardware environment at run-time and choose between programmer pre-defined execution plans. Chroma [18] builds on Spectra but constructs the utility function externally in a more automated fashion. MAUI [11] also reduces the programmer's workload by automating some of the partitioning-process models that its decision engine produces via integer programming techniques.

Before Cloud, opportunistic use of surrogates (untrusted machines) was adopted in [19] and [20]. Slingshot [20] also identifies wireless hotspots as a platform to accommodate the virtual machine capsule. As Cloud Computing and Virtual Machine technologies become mainstream, more researches turned to the Cloud in search of a more secure, accessible and powerful offload platform. OS supported VM migration was introduced in CloneCloud [21]. Calling-the-cloud [22] add a middleware platform that manages an application's execution between the phone and the cloud. A consumption graph is used to model the application. Wishbone [23] looks at the partitioning of sensor network applications in particular and models the decision making process as a integer program. Aimed at reducing the communication costs [9] proposes the concept of Cloudlets, which brings the distant Cloud to the more commonly accessible WiFi hotspots. A dynamic VM synthesis approach is also suggested in [9].

Our research is distinct to all previous work since the applications that we investigate have computation tasks scattered over a group of distributed mobile devices (i.e. a mobile workflow), whereas existing research looks at applications that are implemented on one device only. Our algorithm provides scalable decision making to castaway devices; incorporates authorisation processes; uses clustering techniques to fully exploit the benefit of the cloudlet infrastructure and supports an update-on-event mechanism.

Both energy consumption and execution schedule length are important benchmark metrics for a workflow. Rather than consider only one of these two aspects (in time efficiency [9] [16] [20] [24] and in energy saving [8] [7]) we consider both metrics and the trade-off patterns between the two. A similar analysis on the offload-abilities of tasks is included in [25], but not in any great detail, and also is only based on single smartphone nodes. In our simulation, we carry out a comprehensive analysis of the relation between different characteristics of a workflow and its offload-ability.

III. The Offload Algorithm

In this section, we set the scene by abstracting the mobile workflow and its execution platform into two graphs, and with a simple example demonstrate the impact of an offload action to various interest groups of a workflow. Trade-offs in time and energy of an offload action vary depending on the characteristics of the workflow and the hardware network that carries it. We thus build these variables into our model and construct our objective functions. We then present the algorithms and discuss the design philosophies behind these. Variations of the algorithms are presented and we conclude this section with a discussion on the algorithm's complexity and possible optimisation methods.

A. Preliminaries and problem definition

Two graphs are used in our definition, each annotate the workflow and the hardware network respectively. Firstly, we annotate our mobile workflow as a directed acyclic graph $W = (T, E)$ whose vertices are the set of tasks of the workflow and whose edges are the communications between these tasks. Each task requires a number of instructions to be processed in order to complete its computation, which is given by function *I*. For example $I(t_i)$ gives the number of instructions t_i requires. Since to run the offloaded task on the cloudlet, the executable of the task needs to be transmitted to the cloudlet, we have function $U(t_i)$ to represent the size of t_i's executable. The size of the data carried within each communication call is given by function *D*. Hence we have $D(t_i, t_j)$ to represent the size of the message sent from t_i to t_j.

Our second graph $H = (N, R)$ represents the hardware platform on which our workflow is to be executed. Graph *H*'s vertices are the processing nodes, and its edges represents the data links between these nodes. A processing node $n \in N$ must be either a local smartphone ($n_s \in N_s$) or a cloudlet server ($n_c \in N_c$) but not both, and hence we have $N = N_s \cup N_c \; and \; N_s \cap N_c = \emptyset$. Effectively, this divides the hardware graph *H* into two processing spaces: the *smartphone space* H_s and the *cloudlet space* H_c. Edges within the H_s space interconnect the smartphones together, which in practice is most likely to be carried over the GPRS Core Network unless both phones have established WiFi links. Cloudlet nodes within the H_c space are interconnected via Wide Area Networks (WAN). A data link between the two spaces (i.e. a data link from a smartphone to a cloudlet) is dependent on the

smartphone's location and can be either a 3G or WiFi connection in practice. The bandwidth of each data link varies depending on its carrier, and in our model we annotate function B to obtain the bandwidth property of an edge. For instance we have $B(n_a, n_b)$ which gives the bandwidth between node $n_a$and n_b. We also annotate function S to give the processing speed of each node, for instance, $S(n_a)$ represents the processing speed of node n_a.

The mobile workflow graph W is mapped onto the hardware graph H by two mapping functions: $\alpha : T \mapsto N$ *and* $\beta : E \mapsto R$ to represent the execution plan of the workflow:

$$\alpha(t_i) = n_a \Leftrightarrow task\ t_i\ is\ executed\ on\ node\ n_a$$

$$edge\ e\ joins\ t_i\ to\ t_j \Leftrightarrow \beta(e)\ joins\ \alpha(t_i)\ to\ \alpha(t_j)$$

Before any offload action takes place, our workflow is executed on the smartphone space only, hence:

$$(\forall t)(t \in T \rightarrow \alpha(t) \in N_s)$$

Figure 2 shows an example of a workflow consisting of 3 tasks, and the workflow is originally mapped to the smartphone nodes only:

$$\alpha: T \mapsto N, \alpha(t_1) = n_{s1}, \alpha(t_2) = n_{s2}, \alpha(t_3) = n_{s3}$$

$$\beta: E \mapsto R, \beta(e_1) = r(n_{s1}, n_{s2}), \beta(e_2) = r(n_{s2}, n_{s3})$$

In order to reduce the energy cost of the smartphone space and also to take advantage of the fast processing speed provided by the cloudlet space, our general agenda is to shift the workflow's tasks over to the cloudlet space as much as possible. In our example in Figure 2, task t_2 is offloaded from its *local* smartphone node n_{s2} to cloudlet node n_{c1}, and this changes the mapping functions from W to H as:

$$\alpha': T \mapsto N, \alpha'(t_1) = n_{s1}, \alpha'(t_2) = n_{c1}, \alpha'(t_3) = n_{s3}$$

$$\beta': E \mapsto R, \beta'(e_1) = r(n_{s1}, n_{c1}), \beta'(e_2) = r(n_{c1}, n_{s3})$$

This change effectively expands graph W's destination graph from H's sub-graph H_s to the rest of H and with this expansion comes a series of trade-offs to various interest groups of the workflow:

a) To the user of smartphone node n_{s2}, because task t_2's computation is no longer executed locally, this reduces the energy cost of his handset. Moreover, because the workflow is redirected away from his handset, he also avoids sending and receiving messages to the other handsets which also reduces the energy cost to his handset. The only extra cost incurred from the offload action is that the executables of task t_2 needs to be transmitted to the cloudlet node n_{c1}, which costs energy in this example.

Notice that in a real mobile application, as identified in several papers [26] [11] [23] [27], not all components are suitable for offload. In the most common cases, components which require I/O access must be executed locally on the handset, the same also applies to user interface modules. Thus it is unlikely that a handset can offload all of its duties from the workflow. In such cases those components which are *pinned* on the handset require active connections to be kept between the handset and its neighbours and/or the cloudlet depending on its relation with other tasks in the workflow. Consequently offload becomes a less attractive option to the user.

b) To the users of n_{s1} and n_{s3}, this offload has a negative impact if the distance from it to cloudlet n_{c1} is greater than that to n_{s2}. For instance, consider an enterprise workflow and

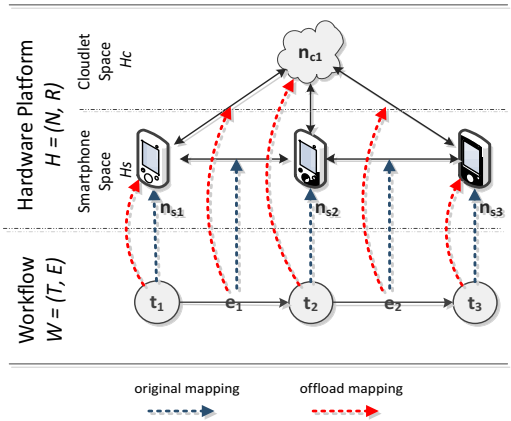


Figure 2: *Offload expands the mapping into the Cloudlet space.*

a time in which both n_{s1} and n_{s2} resides in the same building and are connected through the building's local area network (LAN). Cloudlet n_{c1} however sits externally to this LAN. In such a situation, at least one more network hop is required to complete the communication between t_1 and t_2, which means that n_{s1} must remain active for a longer period of time (with a higher energy cost) in order to confirm a safe exit from the workflow. On the other hand, in a case where n_{s1} is connected to n_{s2} over a long distance network, it is possible that communication from n_{s1} to n_{c1} is shorter than that to n_{s2}, thus the offload is beneficial to the user of n_{s1}.

c) Execution of a typical IT workflow is often constrained by time. While users of individual handset might prioritise energy saving on their phone, the overall time-efficiency of the workflow also needs to be ensured.

From this simple example, we see that managing the trade-offs between *time* and *energy* in various aspects of the workflow is the key element to our algorithm's decision making process. Hence we first capture the time and energy cost both before and after the offload action, and then with these functions we set our objectives to ensure the offload option has at least a positive effect.

1) Time Constraint:

Consider a task t_i which is local to smartphone node $n^l_{t_i}$, we want to see if offloading it to cloudlet node n_c is a beneficial option. We have the time cost before (M^l) and after (M^r) the offload as:

$$M^l(t_i) = \frac{I(t_i)}{S(n^l_{t_i})} + \sum_{(t_j,t_i)\in E} \frac{D(t_j,t_i)}{B\left(n_{t_j}, n^l_{t_i}\right)} + \sum_{(t_i,t_j)\in E} \frac{D(t_i,t_j)}{B\left(n^l_{t_i}, n_{t_j}\right)}$$

$$M^r(t_i,n_c) = \frac{I(t_i)}{S(n_c)} + \sum_{(t_j,t_i)\in E} \frac{D(t_j,t_i)}{B\left(n_{t_j}, n_c\right)} + \sum_{(t_i,t_j)\in E} \frac{D(t_i,t_j)}{B\left(n_c, n_{t_j}\right)} + \frac{U(t_i)}{B(n^l_{t_i}, n_c)}$$

The first term in both functions gives the amount of time task t_i takes to execute on the smartphone and the target cloudlet respectively. The second and third terms are the inbound and outbound communication time costs. Note that n_{t_j} is the node which task t_j is currently assigned to. It can be either task t_j's local smartphone node or a cloudlet node which task t_j is already offloaded to. The last term in the second function is the amount of time it takes to transmit task t_j's executables to n_c.

Our objective is to ensure that the offload action does not delay the workflow's progress. We denote the slack time of task t_i with $M_{t_i}^{slack}$ (the slack time is calculated according to the workflow's critical path) and have our time constraint as:

$$M^r(t_i, n_c) < M^l(t_i) + M_{t_i}^{slack} \quad (1)$$

2) Energy Constraint:

Suppose the current draw on a smartphone node n_s, per second in mA, is $P_c(n_s)$ for computing, $P_i(n_s)$ when it is idle, $P_{ts}(n_s)$ for sending data and $P_{tc}(n_s)$ for receiving data. We have the energy cost on the smartphone before (G^l) and after (G^r) offloading as:

$$G^l(t_i) = \frac{I(t_i)}{S(n_{t_i}^l)} \times P_c(n_{t_i}^l) + \sum_{(t_j,t_i)\in E} \frac{D(t_j,t_i)}{B\left(n_{t_j}, n_{t_i}^l\right)} \times P_{tc}(n_{t_i}^l) + \sum_{(t_i,t_j)\in E} \frac{D(t_i,t_j)}{B\left(n_{t_i}^l, n_{t_j}\right)} \times P_{ts}(n_{t_i}^l)$$

$$G^r(t_i, n_c) = \frac{I(t_i)}{S(n_c)} \times P_i(n_{t_i}^l) + \sum_{(t_j,t_i)\in E \wedge n_{t_j}=n_{t_i}^l} \frac{D(t_j,t_i)}{B\left(n_{t_j}, n_c\right)} \times P_{tc}(n_{t_i}^l) + \sum_{(t_i,t_j)\in E \wedge n_{t_j}=n_{t_i}^l} \frac{D(t_i,t_j)}{B\left(n_c, n_{t_j}\right)} \times P_{ts}(n_{t_i}^l) + \frac{U(t_i)}{B(n_{t_i}^l, n_c)} \times P_{ts}(n_{t_i}^l)$$

The first term in both functions give the amount of energy the smartphone spends whilst the task is being executed. The next two terms are the amount of energy spent receiving and sending data to the neighbouring nodes respectively. Note that if the other end of the communication is on a different node ($n_{t_j} \neq n_{t_i}^l$), no energy is spent at t_i's local node for sending/receive the message.

In order to guarantee that the offload action does not cause the smartphone to consume more energy than its original setting, we set our energy constraint to:

$$G^r(t_i, n_c) < G^l(t_i) \quad (2)$$

For use in our algorithm, we also denote:

$EP_M(t_i, n_c) = it\ satisfies\ the\ time\ constraint\ to\ offload\ t_i\ to\ n_c$

$EP_G(t_i, n_c) = it\ satisfies\ the\ energy\ constraint\ to\ offload\ t_i\ to\ n_c$

B. Algorithm and Design Philosophy

We partition our algorithm into two stages so that is can be implemented on the mobile nodes and the workflow's monitoring server respectively. The first stage (Algorithm 1) is implemented on the smartphones and helps its host to locate the best possible offload point for its tasks according to the environmental parameters it gathers in real-time. For each of its tasks, out of all cloudlets that satisfy both time and energy constraints (if any), it selects the one which gives the largest amount of energy savings as its offload destination. A user has the ability to set a task's property to *isfixed* in order to protect the relevant content from being offloaded. At line 4 $N_c(n_{t_i}^l)$ represents the set of cloudlets that are visible to task t_i's local mobile node at that time.

Algorithm 1 Find the optimal offload node for a task

Input: task object t_i;
Output: if succeeds, return cloudlet node object n_c, otherwise, return *null*;
1: $n_c \leftarrow null$;
2: $g_{max} \leftarrow 0$; // maximum energy saving
3: **if** $t_i.isfixed$ **then return** n_c;
4: **for each** $n_j \in N_c(n_{t_i}^l)$ **do**
5: **if** $EP_M(t_i, n_j) \wedge EP_G(t_i, n_j)$ **then**
6: **if** $(G^l(t_i) - G^r(t_i, n_j)) > g_{max}$ **then**
7: $n_c \leftarrow n_j$;
8: $g_{max} \leftarrow G^l(t_i) - G^r(t_i, n_j)$;
9: **end for**
10: **if** $n_c \neq null$ **then** $t_i.n_{off} \leftarrow n_j$;
11: **return** n_c;

Algorithm 2 Build an offload tree on an offloaded task

Input: offloaded task object t_i;
1: **for each** $t_i' \in T\ s.t.\ t_i \rightarrow t_i' \in E$ **do**
2: **if** $t_i'.n_{off} = null \wedge \neg t_i'.isfixed$ **then**
3: **if** $EP_M(t_i', t_i.n_{off}) \wedge EP_G(t_i', t_i.n_{off})$ **then**
4: $t_i'.n_{off} \leftarrow t_i.n_{off}$;
5: Call **Algorithm 2** with t_i' as input;
6: **end for**

Algorithm 3 Offload a workflow

Input: workflow W;
1: **sort** workloads in set W in topological order;
2: **for each** $t_i \in T$ **do**
3: **if** $t_i.n_{off} = null$ **then**
4: **if** call to **Algorithm 1** with t_i as input returns $non-null$ value **then**
5: Call **Algorithm 2** with t_i as input
6: **end for**

The second stage of our algorithm sits in the server side's workflow engine. Algorithm 3 traverses the list of tasks and communicates with each task's host to see if any offload action is possible. If the host's feedback is positive, then the workflow engine tries to construct an offload tree cluster with that task being the root using Algorithm 2.

The following document some of the algorithm's desired properties that we identified in designing the algorithm:

1) Autonomous Decision Making Ability

Each participating smartphone node should have the ability to make simple offload decisions based on the environment it is currently situated in without prior knowledge or instruction from the server. A mobile wireless data connection, especially when implemented over a cellular network, is prone to connectivity disruption. In such cases, the isolated node should be able to carry on executing its own tasks in an energy-efficient manner. Algorithm 1 is designed to take on such duty.

2) Offload Authorisation

Not all resources on a mobile device are dedicated to a specific workflow. Although an offload action might be beneficial to the overall performance of the workflow, the owner of the device should still be able to have the authority to stop a task and its relevant data to be offloaded. Examples of which include sensitive or private information that the user is not prepared to share; extra financial expenditure for using a faster wireless connection in range, etc. Hence the *isfixed* property as used in Algorithm 1 and Algorithm 2.

This is especially true in choosing the type of wireless connections for the smartphone nodes. In practice, although 3G and WiFi modules can be enabled at the same time on a smartphone, it is normally up to the local operating system to decide which connection is to be used for data transfer tasks. A remote workflow decision engine's role is set to give advice to the user rather than altering the existing settings on the device.

Furthermore, as discussed earlier, some tasks are not suitable to be offloaded. This includes user interface processes, I/O components and processes that are observed by external processes that require the output to be produced on the local node only [26] [11] [23] [27].

3) Task Clustering

Offloading two tasks to the same cloudlet greatly reduces the energy consumption in completing communication tasks. Especially when those tasks belong to different smartphone nodes, clustering essentially eliminates the need to transfer data over a wireless connection between the mobile nodes. In Algorithm 2, once a task has been approved to offload to a cloudlet, we then attempt to exploit the same offload route and offload the same task's leaf tasks to the same cloudlet. Recursive calls to Algorithm 2 expand the offload cluster.

4) Update on Event Mechanism

The outcome of the decision making process depends heavily on the mobile node's real-time environmental parameters. Thus accuracy of this information directly affects the offload's efficiency. However, it is expensive in both time and energy to constantly update the information onto the server [28], especially when no changes have occurred between updates. One solution to this problem is to use the wake-on-event mechanism provided by the mobile's operating systems [29], especially on events like entering a WiFi zone or moving into the range of a Cloudlet as demonstrated in [30].

Our algorithm is designed so that Algorithm 1 is triggered on the handset when significant change has occurred in its network connectivity. Updated information including a new local offload plan is then feedback to the workflow engine.

C. Variations and Optimisation

The algorithm we presented requires both constraints for time and energy, expression (1) and (2), to be satisfied in order for an offload decision to be approved. However, in some cases the workflow would have preference in gaining saving in one metric over the other. For instance, in a business environment, users of the workflow are highly mobile and the handheld device's up time is critical for the users to be able to answer voice calls at all time. A non-time-critical workflow within such an environment has strong preference in saving battery life over execution time. Thus sacrifices in task execution time can be made in order to help reduce the energy consumption on handsets.

Derived from this philosophy to trade-off gains and loses between time and energy, we describe two variations of the algorithm:

1) Minimum Battery Cost

Our first variation prioritises energy saving over time costs. An acceptable time delay $M^{allowed\ delay}$ is added into the time constraint statement; we have the new time constraint as:

$$M^r(t_i, n_c) < M^l(t_i) + M_{t_i}^{slack} + M_{t_i}^{allowed\ delay} \quad (3)$$

This acceptable delay can be either a static value or a dynamic value that is dependent the device's current status (e.g. the current battery level, additional energy saving generated and etc.).

2) Shortest Schedule Length

In some cases, when the ability to re-charge the battery of the smartphone is assured, it is often preferable to take advantage of this opportunity to accelerate the execution of the workflow. In contrast to the first variation, we commit extra energy consumption in exchange for faster execution speed in the second variation. We introduce G^{extra} to the energy constraint and have the modified energy constraint:

$$G^r(t_i, n_c) < G^l(t_i) + G^{extra} \quad (4)$$

In the extreme case where the mobile device is docked to a charging station, we can remove energy constraint EP_G from line 5 in Algorithm 1 and line 3 in Algorithm 2 completely, so that the offload decisions are free from energy constraints.

3) Optimal Condition Expression

Improvements in hardware resources can increase the workflow's offload-ability. However, there is a limit to the hardware's performance. For instance, an individual user's available bandwidth to a WiFi hotspot is often capped. So to send a message of a certain size over this connection takes at least *Data Size / Bandwidth Cap* seconds.

In order to reduce the complexity of our algorithm in real-time, we can use an optimal conditional expression to pre-test a task to see if the time and energy constraints can be satisfied provided that the device is in the best available hardware environments. For instance we can take a bandwidth cap value of 1Mbps into the time constraint and have:

$$M^{r-opt}(t_i, n_c) = \frac{I(t_i)}{S(n_c)} + \sum_{(t_j,t_i)\in E} \frac{D(t_j, t_i)}{1Mbps} + \sum_{(t_i,t_j)\in E} \frac{D(t_i, t_j)}{1Mbps} + \frac{U(t_i)}{1Mbps}$$

If the value given by this expression is greater than the local running time $G^l(t_i)$, this clearly implies that task t_i is not suitable to be offloaded to cloudlet n_c. Increases in cloudlet processing speed also have limited effect on improving the workflow's offload-ability as we discuss in section B of our simulation study. Pre-testing the workflow with this optimal conditional expression can significantly reduce the algorithm's workload at run time.

IV. Experimental Studies

We now present the results of the simulations conducted using our algorithm. Our aim is to find out the impact of our offloading algorithm over workflows of various distinct characteristics on top of different hardware environments. The key parameters of this study are the savings made on the workflow's *total energy consumption* and its *schedule length*. We vary the hardware (e.g. processor speed, 3G/WiFi availability) and software (e.g. computation, executable size) specifications and study their effects on the two metrics. For each environmental setup, we conduct 100 runs of the simulation and use the averages as the experimental result. At the start of each run, our model generates a random workflow which includes 40 independent workloads. Then various parameters are fed into the model to construct a simulation of desired characteristics before we let the offloading algorithm take action. The measured metrics are recorded within each run before and after the offload for analysis.

In the simulation, we expect to see two pairs of metrics affect the offloading decision the most: *communication size and network connectivity*, and *computation size and cloudlet processing speed*. We also profile the energy consumption in our simulation as to what activity it is spent on, and analyse the *energy profile* of the workflow before and after offload.

A. Communication Size and Network Connectivity

In this group of simulations we aim to find out the impact of an increase in communication size over a workflow's offload-ability, and also see if improvements in the wireless connectivity between the smartphone space and the cloudlet domain can help expand the benefits of the offload activity. In order to eliminate the impact from the other critical attributes of a workflow, the computation size, we fix the mean local (smartphone) processing time to $1/1000$ of the mean communication time, so that the offloading decisions in this group of simulations are all only dependent on the workflow's communication size.

An offloaded workflow's communication expense comes from two sources: the process to send the executable to the cloudlet and the re-routed inter-workload communication calls. We look at their impact separately:

1) Executable Size

As shown in Figure 3 increases in a workflow's mean workload executable size derives a decrease in the saving generated by the offload. More WiFi connection reduces the extra cost of transferring the executables and thus generates better offload result. Sending a copy of the executable to the cloudlet server is a procedure solely created to enable the offload action and only makes the offload a more expensive in time and energy.

Like the app stores provided on iOS, Android and Windows Mobile, the concept of an enterprise application store has been widely accepted by the industry and is becoming a common practice in business environments. This eliminates the cost to transfer executables to the server. Similar framework can be found in MAUI [11], which keep a code repository on the server which contain a copy of all executables to overcome this issue.

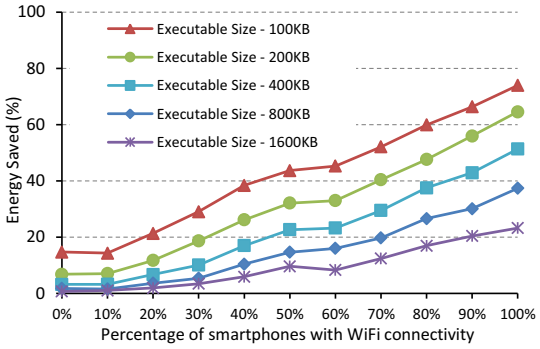
Although the two graphs in Figure 3 look very similar to each other, we notice that at the 0% WiFi connectivity mark, Figure 3.b shows that the savings made in schedule length are mostly zero, whereas Figure 3.a indicates that of the same tests energy savings are positive. One's intuitive assumption would expect the saving in time and energy to be synchronised with each other, and this contradiction seems impossible on first inspection. Furthermore, as in Figure 4.a, out of the 100 runs which the WiFi connectivity was set to zero, the number of runs which occurred saving in energy consumption is more than twice the number of runs with shortened schedule lengths.

In order to understand this result we decomposed this data and found that the extra energy savings come from the tasks that do not reside on the critical path as shown in Figure 4(b). This analysis indicates that to ensure the workflow gets completed no longer that its original schedule length, tasks on its critical path cannot be offloaded with poor network connectivity. However, away from the critical path where the extra communication time created by an offload can be compensated by its slack time, offload is still a feasible choice and helps preserve energy on the mobile nodes.

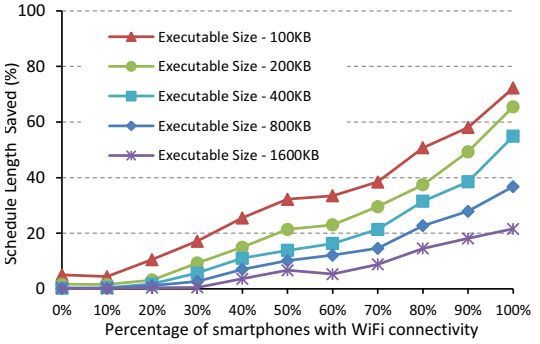
2) Inter-Task Communication Size

It is a shared presumption in many papers [3] [4] [11] that an increase in communication size makes offload less favourable. Our simulation, with increasing executable size brings us to the same conclusion. However, our next set of simulations with increasing inter-task communication size gives us an entirely different picture.

In this group of tests, we exchange the value used for executable size and inter-task communication size in the previous simulation. The remainder of the workflow's attributes stay unchanged. As shown in both plots in Figure 5, the lines are intertwined with each other, which indicate that

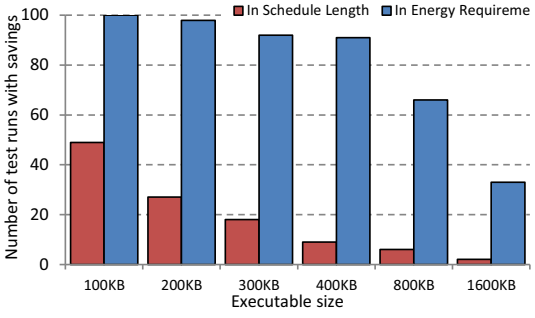


(a)

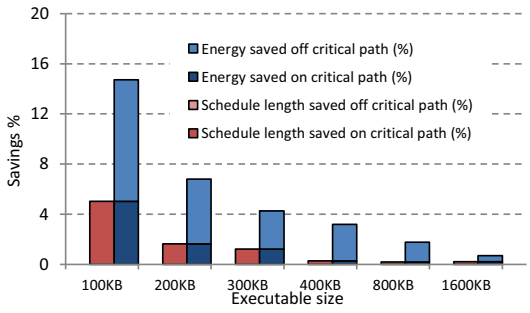


(b)

Figure 3: *As the size of executables increase, fewer savings can be made in the workflow's total energy cost and schedule length (critical path); more WiFi connection makes offload appear more beneficial in both metrics.*



(a)



(b)

Figure 4: *Comparisons of offload savings in energy and schedule length when WiFi connectivity is zero shows that energy savings can be made at workloads that are not on the critical path, even when network resource is poor.*

the increase in inter-task communication size did not have a significant effect on how a workflow is offloaded.

To understand this we need to look at one of the fundamental differences our research has over other work, which is that our experiment is based on a workflow whose tasks are scattered across many different smartphone nodes, rather than all concentrated on one device. In such a case, because the tasks are not all local to the same processor node, every inter-task communication of the workflow would have already had a sizeable cost in both time and energy in the original state. Therefore re-routing these tasks does not necessarily invoke any additional costs.

Figure 6 shows a comparison between two tests with contrasting smartphone to workload ratios. It is very clear in the graph that the workflow that has a higher concentration of workload reacts negatively when its inter-task communication size increases, whereas the other workflow which has half the workload concentration rate shows an opposite trend.

B. Computation Size and Cloudlet Speed

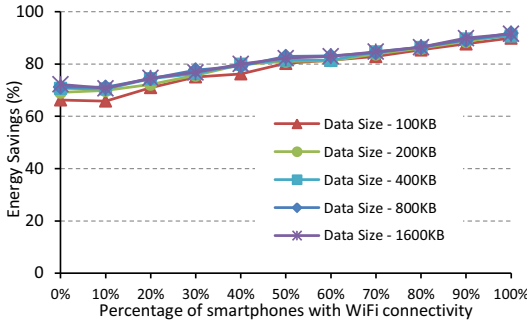
The fast processing speed provided by the cloudlet space helps reduce the execution time of tasks and helps to reduce the overall schedule length of the workflow. Energy wise, although the device might need to be in idle mode whilst waiting for the task to be executed on the cloudlet, the energy cost in idle is much lower than that of computation [29]. Hence we expect a group of cloudlets with higher processing speed to produce better offload gains.

In this group of tests, we set the bandwidth of smartphone-to-cloudlet and smartphone-to-smartphone connections to be the same in order to prevent the result from being influenced by network parameters. Figure 7 shows the variation of savings made in both schedule length and energy consumption with respect to the increase in cloudlet processing speed. We observe that the benefit of offload increases as the cloudlets gets faster. However, both lines fall flat after the smartphone-to-cloudlet gets beyond 1:16.

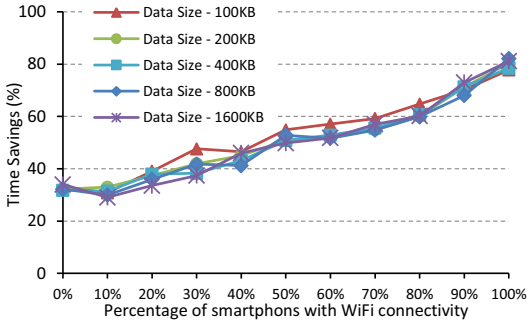
Recall our discussion on an optimal condition expression at the end of the algorithm section. Although we do not apply a cap to the cloudlet's speed in our simulation, the effect from a faster cloudlet is capped to a certain level. In our functions which work out constraint functions (1) and (2), we can see that this is because the functions all follow a reciprocal relation to the processor speeds on the cloudlets - $S(n_c)$. The same also applies to network bandwidth - $B(n_i, n_j)$. This indicates that improvements in hardware environments help increase the offload-ability of workflows but excessive investment is not necessary.

C. Energy Profile

In our tests, as well as seeing the savings made by offload, we are also interested in what activities (computation or communication) the energy was spent on before and after the offload. Figure 8 includes two stacked bar graphs. The one in the background shows the energy distributions of the original workflow and the other, in the foreground, shows the offloaded workflow. The top section of both graphs indicate the share of energy that is spent on communication. The data is grouped so that on the very left is the data gathered from workflows that are offloaded only to take advantage of the fast processing speeds of the cloudlet (i.e. with poor network bandwidth). On the very right are data from workflows that are offloaded only to eliminate communication costs (slow cloudlet speed). We can clearly see that in all groups, the share of energy spent on communication has been increased after offload. This is a clear indication that an offloaded workflow is proportionally more reliant on network connectivity than its original form.



(a)



(b)

Figure 5*: When scattered, increase in workflow's local inter-task message data size does not affect its offload-ability; more WiFi connection makes offload appear more beneficial in both metrics.*

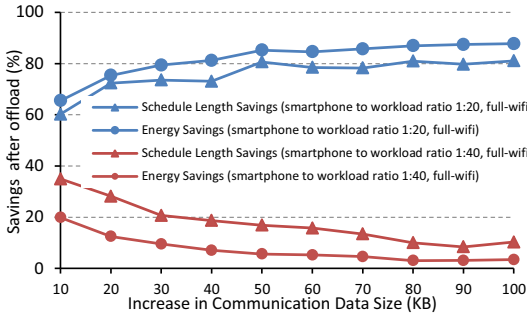


Figure 6: *When workloads are clustered onto a small number of smartphone nodes, workflows with a bigger local communication size produce less energy gain after offload. The contrary applies when workloads are scattered.*

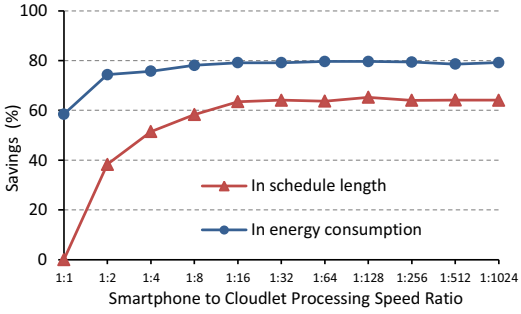


Figure 7: *Improvement in both savings falls flat as cloudlet processing speed increases.*

V. CONCLUSIONS

In this paper we present our approach to managing a mobile workflow over its supporting platform in an energy- and performance-aware manner. With a model which reflects the software and hardware characteristics of the scenario, we present a heuristic algorithm to build and update the offload plan dynamically based on the time and energy constraints of the workflow. Variations of the objective functions are also presented together with optimisation of the algorithm. A series of simulation studies concludes that: 1. when no code repository is available at the server side, a large executable size invariably generates a negative effect on a workflow's offload-ability; 2. large inter-task communication size within a workflow only makes offload less feasible when tasks are concentrated on a small number of smartphones; 3. energy savings can be found easier on workloads that are not on the workflow's critical path, so even when offload is proven not to be preferable by the time constraint, savings can still be made in the workflow's overall energy consumption; 4. the significance of the savings brought about by offload follow a reciprocal relation to the hardware metrics; 5. offloaded workflows are proportionally more reliant on the network.

ACKNOWLEDGMENT

This work is sponsored by the Research Project Grant of the Leverhulme Trust (Grant No. RPG-101).

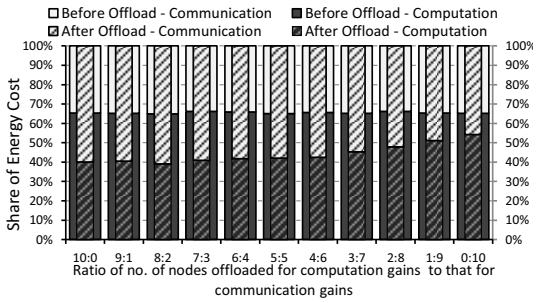


Figure 8: *Offloaded workflows are proportionally more reliant on the network*

REFERENCES

[1] L. Pajunen and S. Chande, "Developing Workflow Engine for Mobile Devices," in *11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)*, 2007, pp. 279-279.

[2] M. Satyanarayanan, "Mobile computing: the Next Decade," in *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services Social Networks and Beyond - MCS '10*, 2010, pp. 1-6.

[3] Gartner Research, "Gartner Reveals Top Predictions for IT Organizations and Users for 2012 and Beyond," *Press Releases*, 2011. [Online]. Available: http://www.gartner.com/it/page.jsp?id=1862714. [Accessed: 04-Mar-2012].

[4] J. A. Paradiso and T. Starner, "Energy Scavenging for Mobile and Wireless Electronics," *IEEE Pervasive Computing*, vol. 4, no. 1, pp. 18-27, Jan. 2005.

[5] K. Pentikousis, "In search of energy-efficient mobile networking," *IEEE Communications Magazine*, vol. 48, no. 1, pp. 95-103, Jan. 2010.

[6] M. Othman and S. Hailes, "Power conservation strategy for mobile computers using load sharing," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 2, no. 1, pp. 44-51, Jan. 1998.

[7] Z. Li, C. Wang, and R. Xu, "Computation offloading to save energy on handheld devices," in *Proceedings of the international conference on Compilers, architecture, and synthesis for embedded systems - CASES '01*, 2001, p. 238.

[8] U. Kremer, J. Hicks, and J. M. Rehg, "Compiler-directed remote task execution for power management," *WORKSHOP ON COMPILERS AND OPERATING SYSTEMS FOR LOW POWER*, 2000.

[9] M. Satyanarayanan, P. Bahl, R. Cáceres, and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14-23, Oct. 2009.

[10] A. P. Miettinen and J. K. Nurminen, "Energy efficiency of mobile clients in cloud computing," in *HotCloud'10 Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, 2010, p. 4.

[11] E. Cuervo et al., "MAUI: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services - MobiSys '10*, 2010, p. 49--62.

[12] A. Gupta and P. Mohapatra, "Energy Consumption and Conservation in WiFi Based Phones: A Measurement-Based Study," in *2007 4th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, 2007, pp. 122-131.

[13] G. P. Perrucci, F. H. P. Fitzek, G. Sasso, W. Kellerer, and J. Widmer, "On the impact of 2G and 3G network usage for mobile phones' battery life," in *2009 European Wireless Conference*, 2009, pp. 255-259.

[14] K. Lee, I. Rhee, J. Lee, S. Chong, and Y. Yi, "Mobile Data Offloading: How Much Can WiFi Deliver?," in *Proceedings of the 6th International COnference on - Co-NEXT '10*, 2010, p. 1.

[15] C. Wang and Z. Li, "Parametric analysis for adaptive computation offloading," in *Proceedings of the ACM SIGPLAN 2004 conference on Programming language design and implementation - PLDI '04*, 2004, vol. 39, no. 6, p. 119.

[16] S. Kim, H. Rim, and H. Han, "Distributed execution for resource-constrained mobile consumer devices," *IEEE Transactions on Consumer Electronics*, vol. 55, no. 2, pp. 376-384, May 2009.

[17] J. Flinn and M. Satyanarayanan, "Balancing performance, energy, and quality in pervasive computing," in *Proceedings 22nd International Conference on Distributed Computing Systems*, 2002, pp. 217-226.

[18] R. K. Balan, M. Satyanarayanan, S. Y. Park, and T. Okoshi, "Tactics-based remote execution for mobile computing," in *Proceedings of the 1st international conference on Mobile systems, applications and services - MobiSys '03*, 2003, pp. 273-286.

[19] J. Flinn, S. Sinnamohideen, N. Tolia, and M. Satyanaryanan, "Data Staging on Untrusted Surrogates," in *USENIX Conference on file and storage technologies (2nd: 2003: San Francisco, CA)*, 2003, pp. 15-28.

[20] Y.-Y. Su and J. Flinn, "Slingshot: Deploying Stateful Services in Wireless Hotspots," in *Proceedings of the 3rd international conference on Mobile systems, applications, and services - MobiSys '05*, 2005, p. 79.

[21] B.-G. Chun and P. Maniatis, "Augmented smartphone applications through clone cloud execution," p. 8, May 2009.

[22] I. Giurgiu, O. Riva, D. Juric, I. Krivulev, and G. Alonso, "Calling the cloud: enabling mobile phones as interfaces to cloud applications," in *Middleware'09 Proceedings of the ACM/IFIP/USENIX 10th international conference on Middleware*, 2009, pp. 83-102.

[23] R. Newton, S. Toledo, L. Girod, H. Balakrishnan, and S. Madden, "Wishbone: profile-based partitioning for sensornet applications," in *Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, 2009, pp. 395-408.

[24] Y. Kun, O. Shumao, and C. Hsiao-Hwa, "On effective offloading services for resource-constrained mobile devices running heavier mobile Internet applications," *IEEE Communications Magazine*, vol. 46, no. 1, pp. 56-63, Jan. 2008.

[25] K. Kumar, "Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?," *Computer*, vol. 43, no. 4, pp. 51-56, Apr. 2010.

[26] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo : a Computation Offloading Framework for Smartphones," in *MOBICASE 2010 IEEE Computer Society*, 2010.

[27] G. Chen, B.-T. Kang, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, and R. Chandramouli, "Studying energy trade offs in offloading computation/compilation in Java-enabled mobile devices," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 9, pp. 795-809, Sep. 2004.

[28] K. Lin, A. Kansal, D. Lymberopoulos, and F. Zhao, "Energy-accuracy trade-off for continuous mobile device location," in *Proceedings of the 8th international conference on Mobile systems, applications, and services - MobiSys '10*, 2010, p. 285.

[29] J. Sharkey, "Coding for life – battery life, that is.," *Google IO Developer Conference, May 2009*, 2009.

[30] E. Shih, P. Bahl, and M. J. Sinclair, "Wake on wireless: An Event Driven Energy Saving Strategy for Battery Operated Devices," in *Proceedings of the 8th annual international conference on Mobile computing and networking - MobiCom '02*, 2002, p. 160.