Saving Energy for Cloud Applications in Mobile Devices using Nearby Resources*

Anas Toma[1], Alexander Starinow[1], Jan Eric Lenssen[2] and Jian-Jia Chen[1]
Department of Computer Science, TU Dortmund University, Dortmund, Germany
E-mail: [1]firstname.lastname@tu-dortmund.de, [2]janeric.lenssen@tu-dortmund.de

***Abstract*—In this paper, we present a middleware to save energy in mobile computing devices that offload tasks to a remote server in the cloud. Saving energy in these devices is very important to prolong the battery life and avoid overheating. The middleware uses an available nearby device called auxiliary server either as a surrogate for the remote one, or as a proxy to pass the data between the mobile device and the remote server. The main idea is to reduce the energy consumption of the communication with the remote server by using a high-speed or a low-power local connection with the auxiliary server instead. The paper also analyzes when it is beneficial to use the auxiliary server based on the response time from the remote server and the bandwidth of the remote connection. The proposed middleware is evaluated using different benchmarks, including commonly used applications in mobile devices, and simulations. Furthermore, it is compared to state-of-the art approaches in this area. The experiments show that The middleware is energy-efficient especially when the bandwidth of the remote communication is relatively low or the server is overloaded.**

I. Introduction

In recent years, there has been an increasing interest in cloud computing. The public cloud market revenue has increased by more than 5 times since 2012 [1]. The cloud provides on-demand computing services over the Internet. It can also be used to improve the performance by providing more powerful computing resources for resource-constrained devices, especially mobile systems such as wearable devices, smartphones, portable medical devices, etc. [9]. Over 80% of global mobile data traffic in 2014 was used for cloud applications and it is expected to reach 90% in 2019 [1]. These applications include mathematical and imaging tools, face and object recognition, image search, optical character readers, games, etc [5, 23]. A mobile device can speed up the execution of complex tasks by executing them remotely on a powerful server in the cloud. In our experiments, it is shown that performing inference with the SqueezeNet CNN [13] on a powerful server (with an NVidia GTX 1080 Ti) speeds up the execution by a factor of 560 compared to executing it on a mobile device (an Odroid XU4 with a Mali-T628 MP6 GPU), using the same OpenCL-based implementation. In addition, the amount of data that needs to be transferred is relatively small, which leads to even more efficient offloading.

However, the efficiency of the mobile cloud computing may be affected by the bandwidth of the connection between the mobile device and the remote server in the cloud, and the response time from the remote server which depends on its workload. Both the bandwidth of the connection and the workload on the server may vary from time to time for many reasons. For instance, there are many mobile medical devices that adopt deep neural networks to perform detection and recognition tasks such as the mobile PAMONO-biosensor [15], which detects nanoparticles like viruses in liquid or air samples. These devices usually offload computation-intensive tasks to a remote server to speed-up the execution. In emergency situations, there will be a high demand on computational power using many devices and the server will be overloaded. Furthermore, the network connection may not be stable or totally unavailable in such situations. Such conditions prolong either the response time from the server or the data transfer time and therefore, the total execution time. Even if this prolongation is affordable, it will result in an additional energy consumption which is an undesirable consequence, especially in mobile systems.

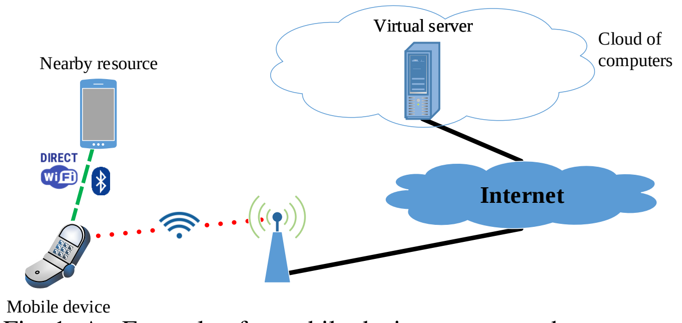


Fig. 1: An Example of a mobile device uses a nearby resource as a surrogate for the remote server to overcome a problem in the connection with the cloud.

In this work, we propose a middleware to save energy in the mobile devices that use cloud resources to speed up the execution of complex tasks. These tasks are executed on the cloud are also called *offloaded tasks*. The middleware exploits the available nearby resources to reduce the energy consumed mainly during data exchange with the remote server in the cloud. Moreover, the middleware can use the nearby resource instead of the remote server to avoid the additional energy consumption due to a low bandwidth connection or an overloaded remote server. Figure 1 shows a mobile device that offloads tasks to a nearby device instead of the server in the cloud in order to overcome the problem of a low bandwidth connection with the cloud. The terms *client* and *auxiliary server* will be used solely when referring to the mobile and the nearby devices respectively.

The main idea is to shorten the time consumed during the data exchange with the remote server by exchanging this data with the auxiliary server. Although this may prolong the idle time on the client during the remote execution of the offloaded tasks, the energy consumption can be still reduced due to the fact that the idle power is less than the transmission and reception power. This approach is especially beneficial when the bandwidth of the connection with the auxiliary resources

*Published in the 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP 2018), Cambridge, UK, March 2018.

is higher than the one with the remote server. However, the response time of both servers should be considered, because usually the remote server is faster than the auxiliary one. Therefore, we also analyze the energy consumption of each scheme and the feasibility of using such a middleware. Our contribution can be summarized as follows:

- A middleware is presented to save energy in mobile devices that use resources in the cloud to improve the performance. The middleware takes advantage of the available nearby resources to reduce the energy consumption of the remote communication by using a local one.
- A nearby resource can be used either as a surrogate for the remote server or as a proxy to pass the data between between the mobile device and the remote server. The middleware selects the most energy saving scheme. Moreover, we provide analysis for the energy consumption of the different schemes above and show when to switch between them.
- The middleware is evaluated using different benchmarks in addition to simulations. The benchmarks include commonly used applications in mobile devices such as face detection, object recognition and body tracking.

II. Literature Review

The concept of executing a computation-intensive task remotely on a powerful server has been widely adopted in the literature under the name of *computation offloading* or *cyber-foraging*. In this section, we provide a summary for the recent studies in the fields of computation offloading and mobile cloud computing. Furthermore, we discuss the limitations of the related approaches.

Abolfazli et al. [3] improve the energy and time efficiency of executing computation intensive mobile cloud applications by offloading computation to cloud-based resources. The work in [22] proposes a middleware to minimize the energy consumption in portable or mobile embedded systems for real-time applications. The tasks are offloaded to a powerful server to reduce the workload on the embedded system which results in a decrease in energy consumption. Most of the available approaches rely on reducing the workload of the mobile device by offloading tasks to powerful servers and take advantage of the idle time during the remote processing of the offloaded tasks to reduce energy consumption. This technique requires a fast response from the remote server and a high-speed connection with it. However, these conditions are not always available, which impedes the usage of cloud services or even makes them inefficient.

A cloud computing model is presented in [12] for mobile devices, especially Internet of Things (IoT) devices. The clients in this model are able to create ad hoc clouds with the help of nearby devices and provide local computing services. The tasks can be offloaded to the nearby devices to improve the performance. This model is useful when the connection to the remote server in the cloud is unavailable. The work in [7] presents a three-layer architecture that is composed of wearable devices, local mobile devices and a remote server in the cloud. The wearable device offloads part of its tasks either to the mobile device or the cloud. This approach is used to maximize the throughput, i.e. the number of the executed tasks, on the wearable device based on genetic algorithm. Both studies in [12] and [7] do not consider the energy consumption of the mobile or the wearable devices.

Zhou et al. [25] propose a framework, called *mCloud*, for code offloading to improve the performance and the availability of mobile cloud computing services in addition to reduce the energy consumption of mobile devices. It consists of mobile devices, nearby cloudlets and a remote server in the cloud. The offloading decision algorithm in this framework selects a wireless medium and appropriate cloud resources based on a cost function. The cost function considers the energy consumption of the wireless channel, i.e. the energy consumption of sending the task and receiving the result, and the task execution time. Considering the energy consumption of just the wireless channel does not guarantee to reduce the total energy consumption of the device, because it consumes additional energy during the idle time spent waiting for the result from the server. Furthermore, this approach uses the nearby resource only as surrogate devices.

Collectively, the existing studies suffer from the following limitations: Conventional computation offloading approaches require relatively very short response time from the server and a high-bandwidth remote connection which are not always available. The approaches that take advantage of the nearby resources consider only improving the performance and do not consider the energy consumption, which is a very important issue in battery-powered devices. The most related approach to ours, i.e., mCloud, does not count the energy consumption of the idle state on the mobile device, which may lead to wrong offloading decisions. Finally, the existing approaches use the nearby resources only as surrogate devices. In contrast to previous works, our approach use the nearby resources either as surrogate or proxy devices. Moreover, we evaluate the total energy consumption of the complete mobile device for each option including the energy of transfer and idle time.

III. Auxiliary Resources to Save Energy

A. System Structure

The system originally consists of a client and a remote server. We present another server between the client and the remote server called auxiliary server. The structure of the system can be described as follows:

- **Client (CL)**: A mobile computing device with limited resources such as wearable devices, smart phones, portable sensors, or any other battery-powered computing device. The client uses a wireless connection to communicate with the remote server.
- **Remote Server (RS)**: A remote virtual server that is more powerful than the client. This server is usually hosted by a cloud of computers or a remote computer.
- **Auxiliary Server (AS)**: A nearby device which is connected directly to the client through Bluetooth, a direct wireless connection (Wi-Fi Direct) or a wireless local area network (WLAN).

B. System Model

We assume that the timing and the power parameters are given based on system setup. Table I summarizes all the notations in this section.

TABLE I: Denotations.

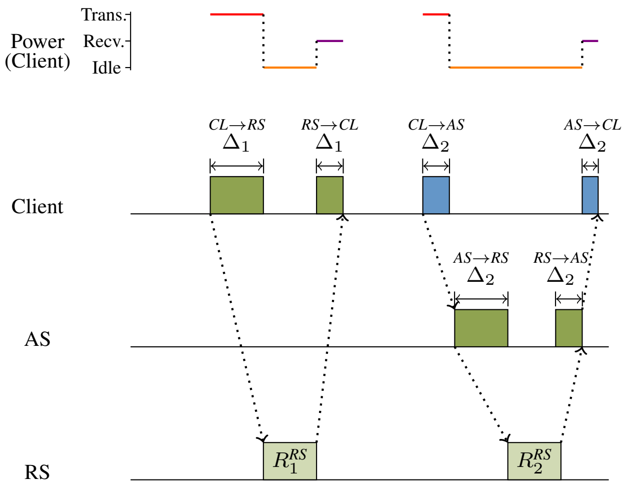
Notation	Description
CL	The client
RS	The remote server
AS	The auxiliary server
τ_i	A task in the system which can be a program, a class, a function, etc
$r_i^{[server]}$	The remote execution time of the offloaded task τ_i on the *server*
$R_i^{[server]}$	The remote response time of the task τ_i offloaded to the *server*
$\overset{[source]\rightarrow[dest.]}{\Delta_i}$	The transfer time of the task τ_i or its result from a *source* to a *destination* device
Ψ_k	The reserved utilization or bandwidth on the server for the client k
$P_{interface}^{state}$	The power consumption of using a communication *interface* during a specific operating *state*
BT	Bluetooth
WiFiD	Wi-Fi Direct
$CL \gg A$, $CL \gg A \gg B$	Offload from the client to the server A or to the server B through the server A respectively
$\overset{[source]\rightarrow[dest.]}{D}$	The size of the transfered data between a *source* and a *destination*
$B_{interface}^{server}$	The bandwidth of the connection to a *server* using a specific *interface*

1) *Task Model:* The task can be a complete program or a part of it such as a function. It can be executed locally on the client or remotely on a server. The code of the task exists on the server or is transferred there just once a time when the connection is established between the client and the server. Then, just the input data of the task is sent to the server for each execution. Therefore, when we say that the task is offloaded to the server, we mean that the input data is sent to the server. Each task τ_i is characterized by the following average timing parameters:

- **Remote execution time** ($r_i^{[server]}$): The execution time of the task on the *server* in the case of offloading, where the *server* can be either the middleware or the remote server, i.e., *server* $\in$ {AS,RS}.
- **Remote response time** ($R_i^{[server]}$): The interval length starting from the time when the task arrives to the *server* until the time when the *server* starts sending back the result. The difference between the remote execution time and the remote response time is explained in Section III-B2.
- **Transfer time** $\overset{[source]\rightarrow[dest.]}{\Delta_i}$: The transfer time of the (data of the) task τ_i or the result of its execution from a *source* device to a *destination* device, where *source* and *dest.* $\in$ {CL, AS, RS}. For example, $\overset{CL\rightarrow AS}{\Delta_3}$ is the transfer time of task τ_3 from the client to the auxiliary server.

Figure 2 shows the timing parameters for an example of the same task. In the first case (shown as task 1), the task is offloaded directly to the remote server. In the second case, (shown as task 2) the task is offloaded to the remote server through the auxiliary server.

2) *Server Model:* If a server is dedicated for one client and ready to start the execution of any offloaded task immediately, then the remote response time is nearly equal to the remote execution time. This is feasible for the auxiliary server. However, a powerful remote server usually serves more than one client and it is impractical to dedicate it for just one client. Suppose that a server is busy with an execution of a task offloaded from a specific client. If the server receives a task from another client, it postpones the execution of the second task until it finishes the execution of the first task. In this case, the remote response time of the second task is longer than its remote execution time and not predictable. Therefore, we need to reserve part of the server's resources (e.g. CPU, GPU, etc.) for each client. In this paper, we adopt the total bandwidth server (TBS) [19] as a resource reservation technique on the server side.


Power (Client)
Trans.
Recv.
Idle
Client
AS
RS
$\overset{CL\rightarrow RS}{\Delta_1}$
$\overset{RS\rightarrow CL}{\Delta_1}$
$\overset{CL\rightarrow AS}{\Delta_2}$
$\overset{AS\rightarrow CL}{\Delta_2}$
$\overset{AS\rightarrow RS}{\Delta_2}$
$\overset{RS\rightarrow AS}{\Delta_2}$
R_1^{RS}
R_2^{RS}
Fig. 2: Timing and power parameters for two offloading cases of the same task.

The TBS has been adopted in the literature to manage the sharing of server's processor by providing a virtual server for each client [21]. Then, the server will be able to serve multiple clients at the same time and prevent the domination of its processor by one client. The server assigns a TBS for each requesting client k with a specific utilization Ψ_k, if possible, where the total given utilization for all the clients should not exceed 100% to preserve the system feasibility [20]. The assigned TBS represents a virtual dedicated server for the client k with a speed equals to $\frac{1}{\Psi_k}$ of the original speed of the server. In this case, the remote response time can be calculated as follows:

$$R_i^{[server]} = \frac{r_i^{[server]}}{\Psi_k} \tag{1}$$

3) Power Model: Let $P^{\textit{state}}_{\textit{interface}}$ denotes the average total power consumption of the client during the usage of a specific communication *interface* and an operating *state*. In this research, we consider **WiFi** and **Bluetooth** as communication interfaces. Their operating states are **idle**, **transmit** and **receive**. However, any other interface can be used such as LTE, 5G, Zigbee, etc. P^{idle} and P^{exc} denote the average total power consumption of the client during the idle and the execution states respectively, without using any communication interface. Execution state means the state when the client executes a task locally at a given frequency and all the communication interfaces are disabled. That is to say, *interface* $\in$ {WiFi, BT, $\emptyset$}, *state* $\in$ {idle, trans, recv, exc}. Figure 2 shows the power consumption of the client for two different offloading schemes.

C. Problem Definition

In this paper, we target the client-server systems, where the client is a mobile, resource-constrained device. It sends the computation-intensive tasks to a powerful remote server through a wireless connection over the Internet in order to be executed remotely. The remote server can be hosted by a remote computer or a cloud of computers. The objective of this research is to reduce the energy consumption of the client device to prolong its battery life by exploiting nearby resources. *The problem is to find a suitable configuration, a nearby resource and a communication medium, that consumes less energy than the direct communication with the remote server.*

D. Auxiliary Server

We provide a middleware for the mobile computing devices that offload tasks to a remote server in the cloud over the Internet using a wireless connection as shown in Figure 3. The tasks are offloaded to the remote server in order to speed up the execution. In some cases, the client may not be able to benefit from the service provided by the remote server efficiently for many reasons as described below:

- Low battery: The battery of the client device may not be powerful enough to use the wireless connection continuously for a long time.
- Low bandwidth: In some places or because of unusual environmental conditions (e.g. Bad weather, earthquakes, etc.), the bandwidth of the connection between the client and the remote server may be low or not available at all. This prolongs data transfer time and then increases the energy consumption on the client.
- Long response time: The remote server may be overloaded or even unreachable due to the very high number of requests, which prolongs the response time. This results in additional energy consumption because of the longer waiting time on the client.

To overcome such problems, the proposed middleware uses a nearby resource as an auxiliary server. The client is connected to the auxiliary server directly through a relatively high-speed connection such as Wi-Fi Direct, or a low-power connection

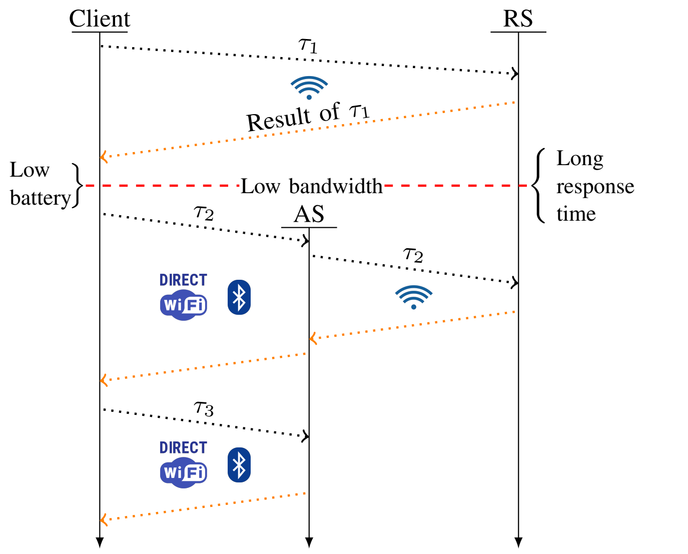


Fig. 3: The usage of the auxiliary server.

such as Bluetooth. Figure 3 shows two schemes to use the auxiliary server:

1) The auxiliary server works as a *proxy*. It passes the offloaded tasks from the client to the remote server and the results inversely.
2) The auxiliary server acts as a *surrogate* for the remote server. It executes the offloaded tasks and returns back the result to the client without passing anything to the server. This scheme is preferable, when the bandwidth of the connection with the server is relatively low or the server is overloaded.

E. Energy Estimation and Decision Making

There are many factors that have influence on the energy consumption of each scheme above. They mainly include the response time of the offloaded task on the server, the bandwidth of the connection, and the power of the communication interface. We provide analysis for the energy consumption of each scheme and discuss the effect of those factors. The following equations estimate the total energy consumption on the client for offloading a given task τ_i according to the two schemes above, in addition to the energy consumption of offloading the task directly to the remote server:

- Total energy consumption by offloading the task to the remote server:

$$\overset{CL\gg RS}{E_i} = \overset{CL\to RS}{\Delta_i} P^{\textit{trans}}_{\textit{WiFi}} + R_i^{RS} \cdot P^{\textit{idle}}_{\textit{WiFi}} + \overset{RS\to CL}{\Delta_i} P^{\textit{recv}}_{\textit{WiFi}} \tag{2}$$

- Total energy consumption by offloading the task to the remote server through the auxiliary one:

$$\overset{CL\gg AS\gg RS}{E_i} = \overset{CL\to AS}{\Delta_i} P^{\textit{trans}}_{\textit{WiFi}} + (\overset{AS\to RS}{\Delta_i} + R_i^{RS} + \overset{RS\to AS}{\Delta_i}) P^{\textit{idle}}_{\textit{WiFi}} + \overset{AS\to CL}{\Delta_i} P^{\textit{recv}}_{\textit{WiFi}} \tag{3}$$

- Total energy consumption by offloading the task to the auxiliary server:

$$\overset{CL\gg AS}{E_i} = \overset{CL\to AS}{\Delta_i} \cdot P^{trans}_{WiFi} + R^{AS}_i \cdot P^{idle}_{WiFi} + \overset{AS\to CL}{\Delta_i} \cdot P^{recv}_{WiFi} \quad (4)$$

Where $\overset{[source]\to[dest.]}{\Delta_i} = \overset{[source]\to[dest.]}{D} / B^{server}_{interface}$, D is the size of the transfered data, and B is the bandwidth of the connection using a communication *interface* during a specific operating *state* as shown in Subsection III-B3. Please note that all other parameters are already presented in Section III-B and summarized in Table I. The Bluetooth connection can be considered as well in Equations (3) and (4) instead of the Wi-Fi connection.

The estimated energy consumption for each scheme in Equations (2), (3) and (4) is composed of three main parts: (a) The energy of transmitting the task, (b) the energy consumed during the idle time while the client is waiting for the result from the server, and (c) the energy of receiving the result. As energy is the integration of power through time, the usage of the auxiliary server may save energy by shortening the time (using WiFiD) or reducing the power (using Bluetooth) of sending and receiving the data. Although this may prolong the idle time on the client, we can still get benefit from the difference between the idle and the sending/receiving power as shown in Figure 2. It shows the power consumption on the client for two offloading schemes using the same task. In the first scheme (related to Equation (2)), the task is offloaded directly to the remote server. In the second scheme (related to Equation (3)), the task is offloaded through the auxiliary server. The middleware evaluates the energy consumption based on Equations (2), (3) and (4) and selects the scheme and the communication interface with the minimum energy consumption for each task τ_i.

Now, we want to find thresholds based on the bandwidth of the connection to the remote server and its response time in order to switch between the three offloading schemes. For notational brevity, let $D = \overset{CL\to RS}{D_i} + \overset{RS\to CL}{D_i}$, which is also equal to both $\overset{CL\to AS}{D_i} + \overset{AS\to CL}{D_i}$ and $\overset{AS\to RS}{D_i} + \overset{RS\to AS}{D_i}$. Suppose that the WiFi communication interface is used and $P = P^{trans}_{WiFi} = P^{recv}_{WiFi}$. Moreover, we omit the subscript *WiFi* in B^{server}_{WiFi}. The usage of the auxiliary server as a proxy saves more energy than offloading directly to the remote server if $\overset{CL\gg AS\gg RS}{E_i} < \overset{CL\gg RS}{E_i}$, which implies that $B^{RS} < \frac{P - P^{idle}_{WiFi}}{P} \cdot B^{AS}$. Therefore, the auxiliary middleware can be chosen to work as a proxy instead of offloading directly to the remote server if $B^{RS} < \alpha \cdot B^{AS}$, where $\alpha = \frac{P - P^{idle}_{WiFi}}{P}$, no matter how much the response time on the remote server changes. Similarly, from Equations (3) and (4), using the auxiliary server as a surrogate saves more energy than using it as a proxy if $B^{RS} < \frac{D}{R^{AS}_i - R^{RS}_i}$ or $R^{AS}_i < \beta + R^{RS}_i$, where $\beta = \frac{D}{B^{RS}}$. Correspondingly, it is more energy-efficient to use the auxiliary server as a surrogate for the remote server when $\overset{CL\gg AS}{E_i} < \overset{CL\gg RS}{E_i}$. From Equations (4) and (2), this implies that $B^{RS} < PD/(\frac{PD}{B^{AS}} + (R^{AS}_i - R^{AS}_i) \cdot P^{idle}_{WiFi})$ or $R^{RS}_i > \frac{PD}{P^{idle}_{WiFi}} \cdot (\frac{1}{B^{AS}} - \frac{1}{B^{RS}}) - R^{AS}_i$.

IV. Experimental Evaluations and Simulations

The proposed middleware was evaluated using different benchmarks and simulations. We use the abbreviation ***AuRes*** to denote our proposed middleware that uses the **Au**xiliary **Res**ources. The energy consumption after using AuRes was evaluated and compared to the case in which the client offloads the tasks directly to the remote server. It was also compared to the state-of-the art approach **mCloud** presented in Section II by considering the energy part of the cost function.

A. Setup

Our approach was evaluated using ODROID-XU4 and a remote server [10]. This device has a Samsung Exynos-5422 processor (Coretex-A15 Quad-Core 2 GHz and Cortex-A7 Quad-core 1.4 GHz), 2GB of RAM and a Mali-T628 MP6 GPU. The same processor is used in Samsung Galaxy S5 mobile device as well [17]. A total of 3 ODROID devices were used as a client, an auxiliary server and a monitoring device to record the power consumption of the client. The client was configured to work as a resource-constrained device by using one CPU running at 500 MHz. The Odroid Smart Power meter [11] was used to measure the total energy consumption of the client device. An external monitoring device was used to read the power consumption from the power meter. To obtain precise measurements, the client sends start and stop signals to the monitoring device before and after the execution of the evaluated benchmark, respectively. through the General-Purpose Input/Output (GPIO) pins in order to avoid any additional overhead during reading. The bandwidth was analyzed by sending and receiving 10 MB packages from the client to the auxiliary and remote servers. Furthermore, simulations were performed to analyze the energy consumption for different bandwidths (1 to 20 Mbps) and response times. The response time on the remote server depends on the number of the served clients and was calculated according to Equation 1.

The benchmarks used in this evaluation are chosen and configured to have different sizes for the exchanged data between the client and the remote server. They are also chosen to have different execution times and varying workloads. These benchmarks are described below:

- **Bodytrack**: Bodytrack is a computer vision application from PARSEC Benchmark Suite [2]. It tracks a 3D pose of a human body with multiple cameras. The used inputs on this benchmark were *simsmall* and *simmedium* with 4 cameras. The simsmall input includes 1 frame and 1000 particles, and the simmedium includes 2 frames and 2000 particles [8]. The output of this benchmark are the coordinates of a tracked body. Bodytrack is commonly used in computer vision applications for mobile devices.
- **Calculating N Prime Numbers**: This benchmark is a simple mathematical calculation that takes a number N as an input and calculates the first N prime numbers. Calculating prime numbers was chosen in this evaluation to represent a heavy mathematical calculation.
- **Arbitrary Matrix Operations**: Mathematical operations on matrices are common operations in modern programs. Therefore, this benchmark performs few arbitrary mathematical operations on two large input matrices. It was written in python using the numpy package.
- **Face detection**: This application takes an image or video stream from a camera as an input. It marks the position of the faces and the eyes in the input image and produces

TABLE II: Benchmarks.

Benchmark	Demand		Description	Data Size (KB)		AuRes Decision
	Computation	Communication		Input	Output	(Offloading Decision, Interface)
MM	Middle	Middle	Bodytrack Small	2500	0.289	$CL \gg AS$, WiFiD
HH	High	High	Bodytrack Medium	5000	0.573	$CL \gg AS$, WiFiD
ML	Middle	Low	Prime Numbers N=45.000	0.006	0.005	$CL \gg AS \gg RS$, BT
HL	High	Low	Prime Numbers N=99.000	0.006	0.005	$CL \gg AS \gg RS$, BT
LM	Low	Middle	Arbitrary Matrix Operations 300x300	1400	720.1	$CL \gg AS$, WiFiD
LL1	Low	Low	Face detection	6.7	21.3	$CL \gg AS \gg RS$, WiFiD
LL2	Low	Low	Squeezenet	54.3	0.011	$CL \gg AS \gg RS$, WiFiD

TABLE III: Power consumption of the ODROID-XU4 running at 500 MHz and using different communication interfaces.

	Average total power consumption (W)			
Interface	Idle	Transmit	Receive	Execute
Wi-Fi (5 GHz)	2.624	3.015	2.974	-
Wi-Fi (2.4 GHz)	2.31	2.807	2.769	-
Bluetooth	2.043	2.273	2.241	-
Disabled (∅)	1.985	-	-	2.037

it as an output. It was chosen as it is a commonly used feature in mobile devices equipped with cameras.

- **SqueezeNet**: One evaluated task is convolutional neural network inference using the SqueezeNet architecture [13]. It performs image classification into the 1000 classes of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [16]. We implemented inference with a trained version of that network in our own OpenCL deep learning inference framework. Therefore, we can evaluate and compare the task on all OpenCL-capable GPUs, which includes several mobile, desktop and server systems.

Every benchmark was run 100 times and the average execution time was recorded for each device, client, auxiliary server and remote server. Table II shows the sizes of the input and output of these benchmarks. The benchmarks are classified and named according to different offloading scenarios. *L*, *M* and *H* letters denote to low, middle and high, respectively. The first and the second positions of the letter represent the computation and the communication demands of the benchmark, respectively. For instance, ML denotes to the benchmark or scenario that requires relatively middle computation and low communication.

B. Results

Table III shows the results of power measurements for the complete ODROID-XU4 board running at 500MHz and using different communication interfaces. As the 5 GHz band consists of up to 25 channels, it is usually used for the normal Wi-Fi connections to avoid the interference with other channels. However, all the Wi-Fi direct devices use the 2.4 GHz band [24]. Table II shows the decision of the AuRes middleware and the selected communication medium between the client and the auxiliary server. The *normalized energy and time saving* using AuRes is equal to 1− (energy and time consumption of using AuRes divided by the energy consumption of offloading directly to the RS). The *normalized data size* of a task is the total size of the exchanged data normalized into the range of [1,20]. This range is just used for representation purposes. The *total Offloading time* of a task is the total time consumed during sending, waiting for the result, and receiving it from the server.

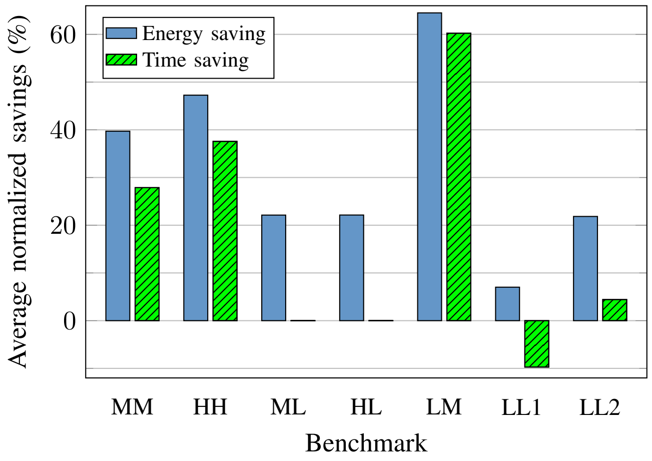


Fig. 4: Average normalized energy and time savings for different benchmarks using AuRes middleware compared to offloading to the RS.

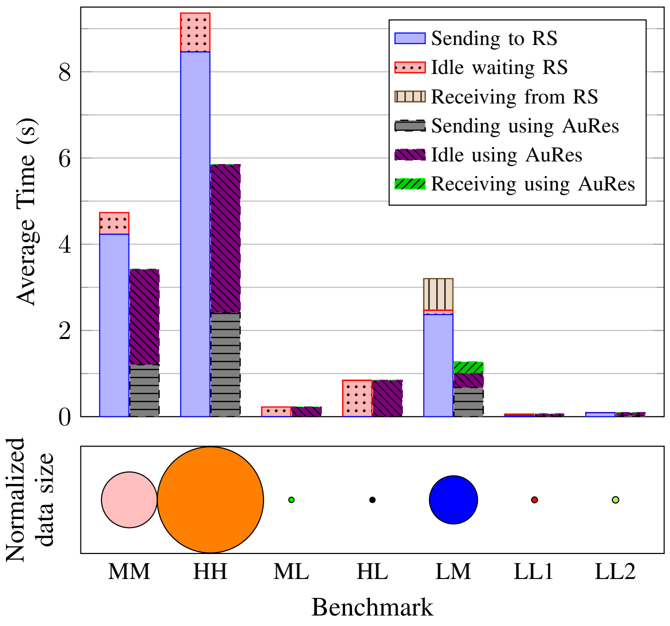


Fig. 5: Sending, idle and receiving time for the benchmarks on the client side by offloading to the RS and using AuRes

Figure 4 presents the average normalized energy and time savings using AuRes for all the considered benchmarks. Although improving the performance is not an objective in

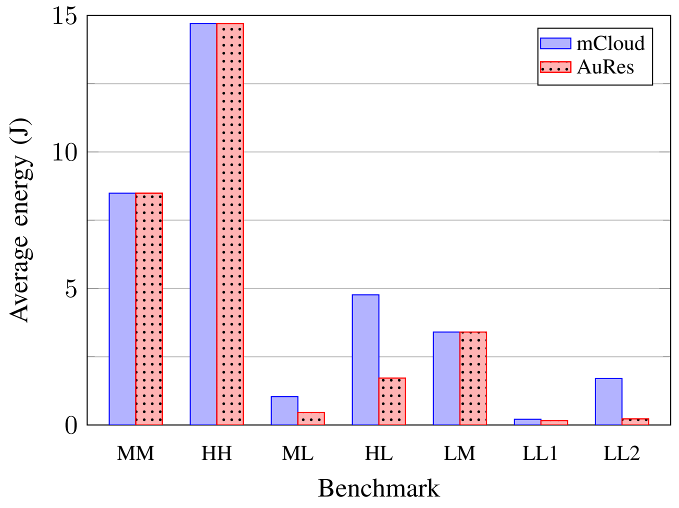


Fig. 6: Energy consumption for different benchmarks using AuRes and mCloud.

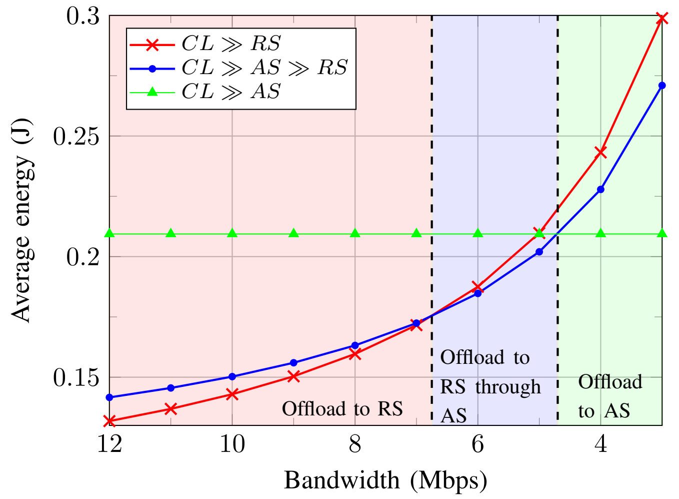


Fig. 7: Average energy consumption of the client by executing the face detection benchmark for different bandwidth values.

this work, we show time savings to evaluate the effect of using the proposed middleware on the total offloading time. Please also note the corresponding Figure 5 which shows sending, idle and receiving time on the client side for all the benchmarks by offloading tasks directly to the server and by using AuRes. It also shows the normalized size of the exchanged data for each benchmark. AuRes is able to save up to 65% of energy and 60% of time for benchmark LM. High energy saving can be achieved for the applications with relatively big size of exchanged data (i.e., the tasks with middle and high communication demands), because AuRes choses the scheme that reduces the transfer time of the data in order to get benefit from the low power consumption during the idle time compared to the relatively high power consumption during data transfer as shown for benchmarks MM, HH and LM in Figure 5. As expected and discussed before, the usage of the auxiliary server may prolong the total offloading time, but it reduces the total energy consumption as shown for benchmarks ML, HL and LL1.

The energy consumption for different benchmarks using AuRes and mCloud is shown in Figure 6. Both approaches have the same energy consumption for the benchmarks MM, HH and LM. The total energy consumption of these benchmarks is dominated by the energy consumption of the communication. However, the energy consumption using AuRes is less than in using mCloud for the remaining four benchmarks. mCloud always prefers to offload the tasks to the auxiliary server through Wi-Fi Direct, because the energy consumption using this communication interface is the minimum over other interfaces for all the benchmarks. This decision may increase the total energy consumption of the tasks with low communication demand (i.e., benchmarks ML, HL and LL1 and LL2), because the energy consumption during the idle time is relatively considerable and the communication energy is not dominant. As discussed in Section II, the selection of the offloading scheme based only on the energy of the communication does not guarantee to achieve the minimum total energy consumption.

To evaluate the effect of the bandwidth of the remote connection on the offloading decision, we fixed the bandwidth of the connection between the client and the auxiliary server to 16 Mbps, and performed the simulation for different bandwidth values for the connection between the client and the remote server. Figure 7 shows the average energy consumption of the face detection benchmark using the three offloading schemes. This task has been chosen because the energy consumption of each offloading scheme has a minimum energy consumption in a different bandwidth range. The energy consumption of offloading to the auxiliary server is constant because we fixed the bandwidth of the local connection. Offloading directly to the remote server has the minimum energy consumption for a relatively high-bandwidth connection (more than 7 Mbps), because the remote response time their is much shorter than on the auxiliary server. As the remote bandwidth decreases, the time and the energy of data transfer increases, which also increases the total energy consumption. For the bandwidth between nearly 4.8 and 7 Mbps, it is more energy-efficient to use the auxiliary server as a proxy, because the client consumes less time and energy by exchanging data with the auxiliary server than with the remote one. The additional energy due to the degradation of the bandwidth will be consumed on the auxiliary server, because it manages the remote data exchange. Although, the idle time on the client may become longer, the total energy consumption usually decreases because the idle power is less than the communication power. For low bandwidth values (less than 4.8 Mbps), it is more beneficial to use the auxiliary server as a surrogate due to the high energy consumption of the remote communication.

To simulate different response times on the remote server, we increased the workload on the server by increasing the number of the served clients. As the number of client increases, the response time on the remote server is also increases. Figures 8 presents the average total energy consumption of the client for the bodytrack medium benchmark and different workloads. This simulation was performed for a high bandwidth of 80 Mbps for the connection to the remote server. Offloading directly to the remote server has the minimum energy consumption for relatively shorter remote response time (i.e. when the number of the tasks executed on the remote server is up to 6). For longer response time, offloading to

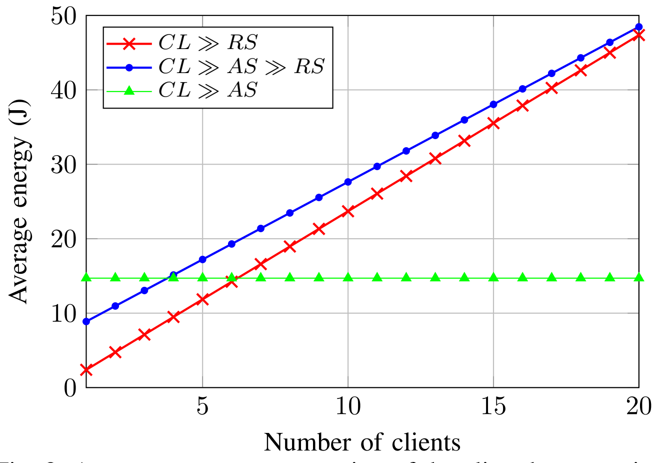


Fig. 8: Average energy consumption of the client by executing the bodytrack medium benchmark for different workloads on the server (i.e., the number of the served clients).

the auxiliary server consumes less energy in total because the idle energy becomes dominant. Therefore, using the auxiliary server as a surrogate is beneficial when the remote server is overloaded.

V. Conclusion and Future Work

In this paper, we propose a middleware that uses the nearby devices to reduce the energy consumption in mobile cloud computing. The middleware can be used as a proxy to pass the data between the mobile device and the remote server in the cloud. It also can be used as a surrogate to for the remote server. The middleware tries to reduce the energy consumed during the remote communication and to overcome the problems of low bandwidth and long response time from the remote server. The proposed approach was evaluated using different applications that are widely used in mobile systems in addition to simulations. The experiments confirmed that the usage of the auxiliary server saves energy compared to the direct offloading to the remote server and a state-of-the art similar approach. They also show that energy saving depends on different factors such as the bandwidth of the connection, the response time from the server and the size of the transfered data. Further research could usefully explore how to exploit the idle time on the client during the waiting of the result of the offloaded tasks. Moreover, it would be interesting to consider the energy consumption of the auxiliary server as well if it was powered by a battery.

VI. Acknowledgements

This work is supported by the German Research Foundation (DFG) as part of the Collaborative Research Center SFB876 (http://sfb876.tu-dortmund.de/), subproject B2.

References

[1] Wikibon. Public cloud market revenue worldwide from 2012 to 2026 (in billion U.S. dollars). Accessed 04 October, 2017. URL https://www.statista.com/statistics/477702/public-cloud-vendor-revenue-forecast/.

[2] The PARSEC benchmark suite. http://parsec.cs.princeton.edu, 2017. Last checked: 12.07.2017.

[3] S. Abolfazli, A. Gani, and M. Chen. Hmcc: A hybrid mobile cloud computing framework exploiting heterogeneous resources. In *2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, pages 157–162, March 2015.

[4] P. Angin, B. Bhargava, and Z. Jin. A self-cloning agents based model for high-performance mobile-cloud computing. In *2015 IEEE 8th International Conference on Cloud Computing*, pages 301–308, June 2015.

[5] A. Bhattacharya and P. De. A survey of adaptation techniques in computation offloading. *Journal of Network and Computer Applications*, 78:97–115, 2017.

[6] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan. Glimpse: Continuous, real-time object recognition on mobile devices. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, SenSys '15, pages 155–168, New York, NY, USA, 2015. ACM.

[7] Z. Cheng, P. Li, J. Wang, and S. Guo. Just-in-time code offloading for wearable computing. *IEEE Transactions on Emerging Topics in Computing*, 3(1):74–83, 2015.

[8] J. P. S. Christian Bienia, Sanjeev Kumar and K. Li. The PARSEC benchmark suite: Characterization and architectural implications. Technical report, 2008.

[9] H. T. Dinh, C. Lee, D. Niyato, and P. Wang. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing*, 13(18):1587–1611, 2013.

[10] Hardkernel. ODROID-XU4, . http://www.hardkernel.com/main/products/prdt_info.php.

[11] Hardkernel. ODROID Smart Power, . http://www.hardkernel.com/main/products/prdt_info.php?g_code=G137361754360.

[12] R. Hasan, M. M. Hossain, and R. Khan. Aura: An iot based cloud infrastructure for localized mobile computation outsourcing. In *2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, pages 183–188, March 2015.

[13] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016.

[14] Y. LeCun, Y. Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[15] J. E. Lenssen, V. Shpacovitch, D. Siedhoff, P. Libuschewski, R. Hergenröder, and F. Weichert. A review of nano-particle analysis with the pamono-sensor. *Biosensors: Advances and Reviews, IFSA Publishing*, pages 81–100, 2017. Publikation.

[16] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[17] Samsung. Exynos 5 octa (5422) mobile processor. http://www.samsung.com/semiconductor/minisite/Exynos/Solution/MobileProcessor/Exynos_5_Octa_5422.html.

[18] M. Satyanarayanan. A brief history of cloud offload: A personal journey from odyssey through cyber foraging to cloudlets. *GetMobile: Mobile Comp. and Comm.*, 18(4):19–23, Jan. 2015.

[19] M. Spuri and G. Buttazzo. Efficient aperiodic service under earliest deadline scheduling. In *Real-Time Systems Symposium*, pages 2 –11, 1994.

[20] M. Spuri and G. Buttazzo. Scheduling aperiodic tasks in dynamic priority systems. *Real-Time Systems*, 10:179–210, 1996.

[21] A. Toma and J. Chen. Server resource reservations for computation offloading in real-time embedded systems. In *the 11th IEEE Symposium on Embedded Systems for Real-time Multimedia (ESTIMedia), Montreal, QC, Canada, October 3-4, 2013*, pages 31–39, 2013.

[22] A. Toma, S. Pagani, J.-J. Chen, W. Karl, and J. Henkel. An energy-efficient middleware for computation offloading in real-time embedded systems. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2016 IEEE 22nd International Conference on*, pages 228–237. IEEE, 2016.

[23] A. u. R. Khan, M. Othman, S. A. Madani, and S. U. Khan. A survey of mobile cloud computing application models. *IEEE Communications Surveys Tutorials*, 16(1):393–413, First 2014.

[24] Wi-Fi Alliance. Does Wi-Fi Direct work on 802.11 a/b/g/n? https://www.wi-fi.org/knowledge-center/faq/does-wi-fi-direct-work-on-80211-abgn.

[25] B. Zhou, A. V. Dastjerdi, R. Calheiros, S. Srirama, and R. Buyya. mcloud: A context-aware offloading framework for heterogeneous mobile cloud. *IEEE Transactions on Services Computing*, 2015.