Comparing REST, SOAP, Socket and gRPC in computation offloading of mobile applications: an energy cost analysis

Carolina Luiza Chamas, Daniel Cordeiro and Marcelo Medeiros Eler
University of São Paulo, São Paulo, Brazil
{carolina.chamas, daniel.cordeiro, marceloeler}@usp.br

***Abstract*—There has been a high concern regarding the energy saving on mobile devices recently, for mobile devices have been performing increasingly complex tasks over time. The computation offloading is one of the most popular techniques used by developers as an effective way of saving energy on mobile devices, which consists on executing complex tasks in external servers with different purposes including save energy. Deciding towards offloading certain tasks requires to understand the influence of the amount of data, amount of computation, and the network profile. Several studies have investigated the influence of different wireless flavours, such as 3G, 4G and wifi, but no study has investigated the influence of the communication choices on the energy cost. Therefore, in this paper, we present an experiment we conducted to evaluate the energy consumption of different communication protocols and architectural styles, namely REST, SOAP, Socket and gRPC, when executing algorithms of different complexities and different input sizes and types. Results show that local execution is more economic with less complex algorithms and small input data. When it comes to remote execution, REST is the most economic choice followed by Socket. Moreover, our data show that computation offloading can save up to 10 time as much energy when compared to local execution for some executions configurations.**

I. Introduction

The most up to date mobile devices provide users with all sort of resources: multi-core processors, gigabytes of RAM, gigabytes of storage, integrated cameras with high resolution, sensors, and a large number of complex applications to perform personal and professional tasks. The increasing popularity of the Internet of Things made the mobile environment the target of many advances, resulting in more complex hardware and software [1]. Naturally, such advances come with a price: the continuous increase in energy consumption.

Unfortunately, the evolution of battery technologies is historically slow [2]. The lithium batteries density improves around 10% per year, but its energy capacity depends on its size. Therefore, only bigger devices would benefit from more battery autonomy that it is available now. However, increasing the device's size goes against the consumer preferences: around 98% of the mobile device users prefer a screen of approximately five inches, but expect each device generation to present more sensors and new peripherals to be incorporated into mobile devices, which requires even more energy [3].

As the mobile devices' size tends to stabilize, reducing energy consumption has become an important concern in this field. Several studies have been performed aiming at identifying how energy consumption can be reduced considering several aspects: use of sensors, network types, programming languages, code obfuscation, apps notifications, and so forth. One of the most investigated strategy to reduce energy consumption in mobile applications is known as computation offloading, which consists on transferring specific computing tasks to an external platform, such as a cluster or a cloud. Beyond reducing energy consumption, offloading computation to an external platform might be done to serve different purposes, such as increase performance and reduce memory and processor usage, for instance.

There are advantages and disadvantages in processing data locally or externally, therefore deciding when to resort to such a technique is a key concern for mobile systems architects and developers. Given a specific goal, the decision of resorting to computation offloading depends on several factors, such as application and network profile, for example. According to Kumar and Lu [4], the energy saved by computation offloading depends on the network, the amount of computation to be performed, and the amount of data to be transmitted. Existing studies thus focus on determining whether to offload computation by predicting the relationships among these three factors. Even though the communication technology or architectural style used to exchange data with the external service plays an important role in this context, approaches found in the literature do not discuss the influence of the communication protocols and architectural style chosen on the energy consumption. Most related work in this field evaluates the influence of wireless communication such as wifi, 3G and 4G, for example.

In this paper, we present an experiment we conducted to evaluate the impact of protocols and architectural style adopted to offload computation in mobile applications, namely SOAP, REST, Socket, and gRPC. To perform such an experiment, we executed classic sorting algorithms with different complexities, input sizes and input types and then compared the amount of energy spent in each case using different communication solutions. By varying the amount of computation to be performed and the amount of data transmitted, we aim at understanding in which contexts one alternative might perform better than the others, including local execution. We believe our findings may help developers and architects to make informed decisions

978-1-5386-2098-4/17/$31.00 2017 IEEE

on the computation offloading of their applications and which communication strategy to use.

This paper is organized as follows. Section II briefly presents computation offloading and energy consumption measurement concepts. Section III discusses the related work. In Section IV, the methodological procedure used in this study is described. Section V presents and discusses the results of our experiments. Finally, Section VI presents the concluding remarks and future directions.

II. Background

Computation offloading consists of transferring tasks to a remote server [5]. The offloading objectives vary: increase performance, minimize energy consumption, increase safety or minimize the use of local resources. Currently, the cloud has become the most popular environment to perform these tasks because of its high availability, low cost and high computational power [6]. The offloading is performed by analysing what should be transferred to the server and what should be processed locally. This analysis process is called offloading decisions [7]. The decisions about performing a task locally or in the cloud can vary according to the stated goals.

Several applications can benefit from computation offloading. Even apparently simple applications may conceal complex algorithms. In a Chess app, for instance, the amount of data to represent the current state of a game is very small, but the amount of computation is very large. In that sense, chess provides an example where offloading is beneficial for most wireless network [4].

In many applications, however, the amount of data that need to be exchanged with external servers may be large. That is why, considering the goal of saving energy, application architects and developers should understand the relationship among the network, the amount of computation to be performed, and the amount of data to be transmitted [4]. Many other parameters can influence the decision, such as the resources available and cost, for example [7], [6]. When it comes to evaluate the influence of the network on the energy consumption, many researchers focus on the type of wireless communication (4G, 3G, wifi). To the best of our knowledge, no study has evaluated the influence of the communications protocol and architectures.

III. Related Work

Several studies have been carried out to understand the feasibility of executing the computation offloading depending on the context and objectives defined. Many tasks performed by applications may not run in a remote environment. A study by [8] showed that an average of 3% of the application methods can be directly executed in a remote environment because they do not depend on direct access to operational system routines, need data synchronization or even depend on other methods. This number can reach up to 47% if the application is refactored in order to facilitate the execution of its tasks remotely, thus it is possible to automate the process of offloading of these methods to the remote environment [9].

In addition to the application structure, there are factors that can influence the decision to perform the computation offloading when the objective is to reduce the energy consumption. Experiments have shown that computing offloading of all possible application tasks can increase the power consumption because the network communication can bring a significant energy cost depending on the amount of data to be processed, the device's processing capacity and the amount of packets sent and received through the network [10], [11], [12]. There has been studies aiming at evaluating the influence of wireless communication [13], [14], [15], such as 3G, 4G and wifi, for example, but we haven't found any research on the influence of the communication protocols and architectures in this context.

IV. Materials and Methods

The computation offloading can be used to increase performance, save resources or even reduce the energy consumption of mobile applications. When it comes to saving energy, one needs to understand the relationship among three important factors: network, amount of computation, and amount of data [4]. We thus design an experiment to understand, with respect the network, the influence of communication protocols and architectural style, namely SOAP, REST, Socket, and gRPC. To experiment this different communication strategies in different contexts we resorted to classic sorting algorithms to set the amount of computation and vary its input size and type to set the amount of data. Therefore, we framed our study by the following research questions:

- *RQ1: Does the amount of data influence the energy consumption?*
- *RQ2: Does the amount of computation influence the energy consumption?*
- *RQ3: Does the communication protocol or architectural style influence the energy consumption?*
- *RQ4: Can computation offloading save energy?*

A. Materials

On the network side, we chose SOAP, REST, Socket and gRPC because they are well establish ways of interoperating data among systems. With respect to the amount of computation factor, we selected classic sorting algorithms with well known implementations and complexities. The table I presents the set of sorting algorithms selected for this investigation. This study used the classical sorting algorithms: Bubble Sort and Selection Sort, considered simple algorithms; and Heap Sort, considered more sophisticated [16]. It is reasonable to think many apps use such types of algorithms or algorithms with similar complexities. Regarding the amount of data, well vary the input size and type of the vectors sorted by the algorithms.

On the client side, we implemented a native Android application to execute the algorithms presented in table I. The mobile device used is a Samsung Galaxy S5 with Android 6.0.1 (Marshmallow), 2800mAh 3.85V 10.78Wh battery, 2 processors, Octa-Core, 2.1Ghz Quad-Core ARM Cortex- A15

TABLE I
SORTING ALGORITHMS COMPLEXITIES

Algorithm	Complexity Best Case	Complexity Worst Case
Bubble Sort	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$
Selection Sort	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
Heap Sort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$

and 1.5Ghz Quad-Core ARM Cortex-A7 and 2Gb of LPDDR3 memory. In this app, the user can set the following parameters:

- Input size: 1000, 10000, 100000.
- Input type: int, float or object. This object is instance of a class that declares two strings and two integers. One of the strings has 12 characters while the other has 16.
- Case: best or worst case scenario. When the best case is selected, the content of the vector to be submitted to the sorting algorithm is created in an ascending order. When the worst case is selected, the vector's content is created in a descending order.
- The sorting algorithm: Bubble Sort, Selection Sort and Heap Sort.
- Type of execution: local or remote, which can be SOAP, REST, Socket, or gRPC.

On the server side, we implemented the sorting algorithms using the Java language. The communication between the mobile device and the server is done via standard 802.11 connection (known as Wi-Fi or Wireless). The server we use to host the services with the sorting algorithms has the follow configuration: Windows 7 operating system, 2.20Ghz Intel Core 2 Duo T6600 CPU per core and 4GB of memory.

The energy consumption was measured using the Android Battery Manager interface, which is an internal measurement that provides ways to get information about the system's battery level [17].

B. Methods

Given the large amount of combinations of varying factors, we automated the process of executing the experiments. The overall process works as follows:

1 Charge device until its battery level reaches 100%.
2 Choose an execution configuration by setting the application parameters (algorithm, case, input size, input type, remote/local) - see Section IV-A.
3 Create a vector to be sorted based on the execution configuration (case, input size and input type)
4 Execute the algorithm to sort the vector created in step 3 (locally or remotely, depending on the parameters set)
5 Collect battery level
6 If battery level has been consumed in 1%, go to Step 7, otherwise Step 3
7 If battery level is less than 10%, go to Step 1, otherwise Step 2

Notice that we don't collect the energy consumed from a single execution of each execution configuration. Instead, we repeat the process of creating the vector to be sorted and executing the algorithm until the battery level has been consumed in 1%. Our initial idea was to run the algorithm a fixed number of times and then collect the battery level consumed. However, the Android Battery Manager interface can only return the battery level using an integer type. Only when 1% of battery is consumed it decreases the battery level in 1%. That way, we execute the algorithms as many time as necessary to consume 1% of the battery, which means the more the number of executions, the less the battery consumed by that execution configuration. According to the properties of the Galaxy S5 device used, 1% of battery is equal 388.1J, hence it is possible to calculate the average energy consumed by each execution by dividing 388.1J by the number of executions.

The executions for a particular configuration stop when the battery level is consumed in 1%. Before selecting a different execution configuration by setting the app parameters, the battery level is checked again. If it is below 10%, the device's battery is fully charged (Step 1). If the battery level is above 10%, a new execution configuration can be chosen and executed.

The energy consumed when executing the same configuration can vary in both local execution and remote execution. Local execution can be influenced by operational system kernel or other routines running in the background, for example. Remote execution is mostly influenced by network performance. Therefore, we repeat the experiments execution 10 times to obtain reliable data with a small margin of error the influences of factors that are not under our control.

There are factors that can influence the results of our experiments. We have only executed experiments on a single device. In that case, results might be influenced by a particular device architecture. Furthermore, we have not randomized the execution of each configuration so they could be executed considering different battery levels. For instance, due to the non-linearity of lithium batteries [18], executing a configuration execution with battery level in 100% might be different if the battery level were 30%.

V. RESULTS

This section presents the results of the experiments conducted to answer the research questions presented in Section IV. The large amount of variables that can influence energy consumption makes it possible to perform several analysis by combining different configurations. However, given the limited space, in this paper we only present the raw data we collected on our experiments execution. Table II shows the results of our experiments considering the average of 10 trials. Each row of this table presents the results for a particular configuration execution. Columns 1 and 2 shows the parameters related to the amount of data to be processed: input size and type. Columns 3 to 5 shows the parameters that refers to the amount of computation to be performed: algorithm, best or worst case, and complexity. Columns 6 to 10 shows the amount of energy consumed when the algorithm is executed locally, using SOAP, REST, Socket or gRPC, respectively. Values in Columns 6 to 10 represents the amount of energy

consumed, in average, by a single execution of that particular configuration using that specific communication strategy. This values was calculated by dividing 388.1J by the number of executions of that configuration that take to consume 1% of battery (see Section IV-B).

The columns highlighted in bold indicates the communication strategy that consumed less energy for that particular configuration. In fact, the table is sorted by the execution that consumed less energy. If you follow the values in bold from top to down on the table, you'll notice they appear in an ascending order. This is useful to observe whether there are clear influencing factors. Missing values on local execution column means it was not possible to execute that algorithm locally.

A. RQ1 - Does the amount of data influence the energy consumption?

It is clear that the amount of data plays an important role in this context. Following the results of the most economic executions, indicated by the columns in bold, the column input size is almost sorted in an ascending order, from 1000 to 100000. Notice that for all configurations with 1000 elements the local execution was the most economic, except for a few object input types, for which REST was the most economic execution. For a small amount of data, the overhead of the communication was bigger than executing everything locally. In some cases the difference is quite big.

For 10000 elements, there is still a predominance of local and REST amongst the most economic options, with Socket being the most economic occasionally. When a configuration of 10000 elements is more economic than a 100000 execution, considering the most economic choices, it usually refers to a primitive value (float or int). When the amount of data increases due to the use of a more complex type, the energy consumption also increases.

We can also notice that the amount of energy consumed increases considerably from 1000 elements to 10000 and then to 100000, even by the most economic choices. The Table III shows the differences in the energy consumption between the different input sizes using integer type in locale executions as example. Analysing the variation of energy consumption among the different input sizes, algorithms $\mathcal{O}(n)$ and $\mathcal{O}(n \log n)$ had a stable variation of energy, whereas the algorithms $\mathcal{O}(n^2)$ presented a great variation. While the algorithms $\mathcal{O}(n)$ and $\mathcal{O}(n \log n)$ had a variation of a maximum of 10 times more between 1,000 - 10,000 elements, the $\mathcal{O}(n^2)$ algorithms consumed approximately 88 to 110 times more energy.

B. RQ2: Does the amount of computation influence the energy consumption?

The amount of computation is given by the algorithm and its complexity, which depends on the best or worst case. Notice that for almost all executions with 1000 elements the local execution was more economic, including three configurations using objects, but only for algorithms with linear complexity ($\mathcal{O}(n)$ and $\mathcal{O}(n \log n)$). For quadratic algorithms, however, the local executions of 1000 of object elements were not the most economic executions.

We can see the same pattern for 10000 elements. Local execution is usually more economic than remote execution when the algorithm complexity is linear. The combination of 10000 elements, object types and quadratic algorithms make it more energy consuming than configurations using 100000 elements, but primitive types and linear algorithms. Clearly, the quadratic algorithms consumes much more battery than linear algorithms, even for different input sizes.

Considering algorithms with complexity, the Bubble Sort, in the best case, consumed less energy than the Insertion Sort for the integer type with all input sizes. Analysing with other data types such as float and object, this behaviour changed - Bubble Sort consumed more energy than the Insertion Sort. In special, considering input size of 100,000 elements and input type object, the difference is only 2 executions between them for 1% of battery. The results indicated that Bubble Sort consumed more energy among the worst-case algorithms with all inputs and data types, failing to execute with vector of 100,000 elements and object type, and spending 388.1J to execute a vector with 100,000 float type elements. Considering the $\mathcal{O}(n \log n)$ algorithms, Heap Sort consumed more energy in almost all cases than Merge Sort, in all cases, input types and sizes. Considering the $\mathcal{O}(n^2)$ algorithms, Insertion Sort, in the worst case, was the one that consumed less energy, except for two situations compared to Selection Sort.

C. RQ3: Does the communication protocol or architectural style influence the energy consumption?

Table II shows that the communication protocol or architectural style has a huge impact on the energy consumption. The firs thing to notice here is that, except in two instances, local execution was more economic than remote execution for linear algorithms, considering all input sizes and types. Considering the cases in which the remote execution was more economic than local execution, REST and Socket were the more economic all the time. Surprisingly, gRPC was never the most economic configuration.

If we rule out the local execution, REST was the more economic 31 times, Socket 16 times, SOAP 5 times and GRPC 2 times. REST and SOAP are the most economic choices for computations offloading in this context, with REST being clearly ahead of all other options. Socket tends to consume less energy with primitive data types, in contrast REST tends to consume less energy with object data type. In some configurations, the difference between the energy consumptions may be from 2 - 5 times as much for execution, which can cause quite an impact on the overall energy consumption. However, with small input sizes, primitive types and linear complexities, the difference is usually quite small.

D. RQ4: Can computation offloading save energy?

Local execution was more economic than remote execution in 31 occasions against 20 on the remote side. When the

TABLE II
RESULTS FOR LOCAL AND REMOTE EXECUTIONS

INPUT SIZE	INPUT TYPE	ALGORITHM	CASE	COMPLEXITY	LOCAL	REST	SOAP	SOCKET	GRPC
1000	Int	Bubble Sort	Best	$\mathcal{O}(n)$	**0,0066**	0,0971	0,1957	0,4731	0,1535
1000	Int	Heap Sort	Worst	$\mathcal{O}(n \log n)$	**0,0084**	0,0979	0,1281	0,2462	0,1173
1000	Int	Heap Sort	Best	$\mathcal{O}(n \log n)$	**0,0120**	0,1105	0,1749	0,2819	0,2023
1000	Int	Selection Sort	Worst	$\mathcal{O}(n^2)$	**0,0144**	0,0808	0,1116	0,2373	0,1345
1000	Int	Bubble Sort	Worst	$\mathcal{O}(n^2)$	**0,0273**	0,1455	0,1888	0,1995	0,1032
1000	Int	Selection Sort	Best	$\mathcal{O}(n^2)$	**0,0286**	0,1073	0,1723	0,3590	0,2247
10000	Int	Bubble Sort	Best	$\mathcal{O}(n)$	**0,0562**	0,3842	5,2375	0,5723	0,6103
1000	Float	Heap Sort	Worst	$\mathcal{O}(n \log n)$	**0,0705**	0,1379	0,2245	0,2734	0,1442
1000	Float	Bubble Sort	Best	$\mathcal{O}(n)$	**0,0788**	0,1221	0,1796	0,2532	0,0909
1000	Float	Heap Sort	Best	$\mathcal{O}(n \log n)$	**0,0846**	0,1164	0,1504	0,2004	0,1635
10000	Int	Heap Sort	Worst	$\mathcal{O}(n \log n)$	**0,0852**	0,5927	0,6151	0,6963	1,0327
1000	Float	Selection Sort	Worst	$\mathcal{O}(n^2)$	**0,0859**	0,1101	0,1304	0,2528	0,1669
1000	Float	Selection Sort	Best	$\mathcal{O}(n^2)$	**0,0908**	0,1225	0,1044	0,3539	0,1566
10000	Int	Heap Sort	Best	$\mathcal{O}(n \log n)$	**0,0989**	0,5654	0,6684	0,5826	1,5551
1000	Float	Bubble Sort	Worst	$\mathcal{O}(n^2)$	**0,1035**	0,1103	0,1515	0,3035	0,1214
1000	Object	Bubble Sort	Best	$\mathcal{O}(n)$	**0,1881**	0,4674	1,1716	3,0535	0,7943
1000	Object	Heap Sort	Best	$\mathcal{O}(n \log n)$	**0,2069**	0,4666	0,9995	0,7987	1,2325
1000	Object	Heap Sort	Worst	$\mathcal{O}(n \log n)$	**0,2785**	0,5862	1,3291	0,9457	1,3672
100000	Int	Bubble Sort	Best	$\mathcal{O}(n)$	**0,5633**	11,3149	5,9985	5,1541	11,2820
1000	Object	Selection Sort	Worst	$\mathcal{O}(n^2)$	0,5904	**0,5746**	1,2455	0,7847	1,5713
1000	Object	Selection Sort	Best	$\mathcal{O}(n^2)$	0,6567	**0,5887**	1,3560	1,1486	1,7529
10000	Float	Heap Sort	Best	$\mathcal{O}(n \log n)$	0,7628	0,8458	0,9007	**0,6607**	2,0516
10000	Float	Bubble Sort	Best	$\mathcal{O}(n)$	**0,6806**	0,9037	1,0447	0,7419	1,1520
1000	Object	Bubble Sort	Worst	$\mathcal{O}(n^2)$	1,1236	**0,6971**	1,0281	3,3813	0,9195
10000	Int	Selection Sort	Best	$\mathcal{O}(n^2)$	0,8400	0,7700	1,1201	**0,7021**	1,5014
10000	Float	Heap Sort	Worst	$\mathcal{O}(n \log n)$	**0,7362**	1,0345	0,7587	0,8494	1,1146
10000	Int	Selection Sort	Worst	$\mathcal{O}(n^2)$	**0,8761**	1,4235	1,1707	1,3544	1,1711
10000	Float	Selection Sort	Best	$\mathcal{O}(n^2)$	1,9137	0,9990	0,9197	**0,8871**	1,9734
100000	Int	Heap Sort	Worst	$\mathcal{O}(n \log n)$	**0,9592**	8,1453	6,4900	3,9846	7,8089
10000	Float	Selection Sort	Worst	$\mathcal{O}(n^2)$	2,3450	**0,9733**	1,0067	1,0259	1,1051
100000	Int	Heap Sort	Best	$\mathcal{O}(n \log n)$	**1,0054**	4,9058	7,4100	5,1201	11,1523
10000	Int	Bubble Sort	Worst	$\mathcal{O}(n^2)$	2,3986	1,6480	1,7885	**1,1022**	1,8684
10000	Float	Bubble Sort	Worst	$\mathcal{O}(n^2)$	4,8271	**1,7052**	1,9241	1,8161	1,8152
10000	Object	Bubble Sort	Best	$\mathcal{O}(n)$	**2,0491**	6,1701	9,5123	25,8733	10,7144
10000	Object	Heap Sort	Best	$\mathcal{O}(n \log n)$	**2,3839**	3,7863	13,0235	7,1211	19,2446
10000	Object	Heap Sort	Worst	$\mathcal{O}(n \log n)$	**2,6082**	53,1084	14,9990	6,7561	11,1523
10000	Object	Selection Sort	Best	$\mathcal{O}(n^2)$	47,3293	**5,0370**	10,3955	7,4065	16,3067
100000	Float	Heap Sort	Best	$\mathcal{O}(n \log n)$	6,9552	7,9610	12,2928	**5,5922**	10,8711
10000	Object	Selection Sort	Worst	$\mathcal{O}(n^2)$	44,3543	**5,7710**	15,4622	10,0284	10,8408
100000	Float	Heap Sort	Worst	$\mathcal{O}(n \log n)$	8,4738	10,3493	7,4396	**6,1408**	12,7246
100000	Float	Bubble Sort	Best	$\mathcal{O}(n)$	**6,3832**	10,3493	7,3783	6,6799	15,2624
10000	Object	Bubble Sort	Worst	$\mathcal{O}(n^2)$	84,3696	**9,7146**	15,6492	10,9943	15,6132
100000	Int	Selection Sort	Best	$\mathcal{O}(n^2)$	66,9138	13,4757	16,9476	**11,9876**	39,6920
100000	Float	Selection Sort	Best	$\mathcal{O}(n^2)$	121,2813	**16,0041**	20,2135	22,1771	33,9588
100000	Object	Bubble Sort	Best	$\mathcal{O}(n)$	**20,0052**	70,5636	104,8919	242,5625	166,3286
100000	Object	Heap Sort	Best	$\mathcal{O}(n \log n)$	**22,3046**	44,8823	133,8276	77,6200	86,2444
100000	Float	Selection Sort	Worst	$\mathcal{O}(n^2)$	194,0500	31,8115	31,3616	**22,9645**	35,6874
100000	Int	Selection Sort	Worst	$\mathcal{O}(n^2)$	88,2045	29,5133	**23,3795**	34,6518	30,5247
100000	Object	Heap Sort	Worst	$\mathcal{O}(n \log n)$	**23,6646**	199,2946	129,3667	57,9254	77,6200
100000	Int	Bubble Sort	Worst	$\mathcal{O}(n^2)$	242,5625	47,9136	**36,2710**	46,1987	56,5979
100000	Float	Bubble Sort	Worst	$\mathcal{O}(n^2)$	388,1000	50,7320	51,7467	**37,6796**	86,2444
100000	Object	Selection Sort	Best	$\mathcal{O}(n^2)$	-	**215,6111**	251,9835	242,5625	232,86
100000	Object	Selection Sort	Worst	$\mathcal{O}(n^2)$	-	**250,3870**	388,1	275,9824	277,2143
100000	Object	Bubble Sort	Worst	$\mathcal{O}(n^2)$	-	**388,1000**	**388,1000**	**388,1000**	**388,1000**

remote execution is more economic, and specially for complex algorithms and big input sizes, the amount of energy saved by remote execution is worthy. Although there are cases in which the energy saved would be around 3%, it can save up to 10 times as much energy. Consider, for example, the configuration 100000 elements, Float, Bubble Sort, Worst case, $\mathcal{O}(n^2)$. The energy consumed by local execution was 388,1, while Socket consumed 37,6796. Therefore, the answer to this research question that motivates continuous work in this area is that computation offloading can indeed save a lot of energy in mobile applications. In this particular analysis we have extended the number of sorting algorithms used by including the Merge Sort and Insertion Sort algorithms.

VI. CONCLUDING REMARKS

Energy saving strategies have been increasingly investigated in the context of mobile applications. In particular, the com-

TABLE III
ENERGY CONSUMPTION COMPARATIVE OF DIFFERENT INPUT SIZES

Algorithm	Case	Comparative 1,000 x 10,000 (times higher)	Comparative 10,000 x 100,000 (times higher)
Bubble Sort	Best	8.5	10.0
Heap Sort	Best	8.3	10.2
Merge Sort	Best	7.6	8.6
Insertion Sort	Best	9.0	9.3
Selection Sort	Best	29.3	79.7
Bubble Sort	Worst	87.8	101.1
Heap Sort	Worst	10.1	11.3
Merge Sort	Worst	9.0	11.1
Insertion Sort	Worst	58.8	109.4
Selection Sort	Worst	60.8	100.7

putation offloading strategy requires architects and developers to understand the relationship and influences of the amount of data, the amount of computation and the network profile in energy consumption. As far as we know, no research have been published so far comparing the influence of the communication protocols and architectural styles on energy saving during computation offloading. Therefore, in this paper we presented an experiment we conducted to show the energy spent to execute sorting algorithms with different input sizes and types when executed locally, using REST, SOAP, Socket and gRPC. Results show that the input size and type plays a huge influence on the energy consumption, as well as the complexity of the algorithms. With respect to the communication choices, REST is by far the most economic option, followed by Socket. SOAP and gRPC were more economic in very rare occasions.

Microservices architecture is a trend on application implementations nowadays, specially because it has been shown as a desired resilience feature that provides a fast software delivery having smaller independent services. Since REST is the emerged architectural style to handle that type of application [19], this type of application tend to be more energy efficient.

Remote executions were the most economic choice overall, but only for small input sizes and types, and linear algorithms. With more complex algorithms and larger inputs, computation offloading can save up to 10 times as much energy than local execution. Even though some of the results of our experiments might be obvious in the sense that larger input sizes, more complex types and algorithms tend to consume more energy, it is important to quantify such results to provide architects and developers with data so they can make more informed decisions. On top of that, figuring out which are the most economic choices for communication can also help developers to decide which technology to use in computation offloading and hence increase the energy saved.

As future work, we intend to replicate this experiments using different algorithms, input types and devices. We intend to use real world applications to offload their algorithms to evaluate in which conditions the offloading is worthy and with which communication protocol or architectures.

REFERENCES

[1] M. Aazam and E.-N. Huh, "Fog computing: The cloud-IoT/IoE middleware paradigm," *IEEE Potentials*, vol. 35, no. 3, pp. 40–44, 2016.

[2] S. Tarkoma, M. Siekkinen, E. Lagerspetz, and Y. Xiao, *Smartphone energy consumption: modeling and optimization*. Cambridge University Press, 2014.

[3] M. Halpern, Y. Zhu, and V. J. Reddi, "Mobile CPU's rise to power: Quantifying the impact of generational mobile CPU design trends on performance, energy, and user satisfaction," in *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 64–76.

[4] K. Kumar and Y. H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *Computer*, vol. 43, no. 4, pp. 51–56, 2010.

[5] D. De, *Mobile cloud computing: architectures, algorithms and applications*. CRC Press, 2016.

[6] A. Fahim, A. Mtibaa, and K. A. Harras, "Making the case for computational offloading in mobile device clouds," in *Proceedings of the 19th Annual International Conference on Mobile Computing & Networking*, ser. MobiCom '13. New York, NY, USA: ACM, 2013, pp. 203–205.

[7] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mobile Networks and Applications*, vol. 18, no. 1, pp. 129–140, 2013.

[8] A. Saarinen, M. Siekkinen, Y. Xiao, J. K. Nurminen, M. Kemppainen, and P. Hui, "Can offloading save energy for popular apps?" in *Proceedings of the seventh ACM international workshop on Mobility in the evolving internet architecture*. ACM, 2012, pp. 3–10.

[9] Y.-W. Kwon and E. Tilevich, "Reducing the energy consumption of mobile applications behind the scenes," in *2013 IEEE International Conference on Software Maintenance*, Sep 2013, pp. 170–179.

[10] L. Corral, A. B. Georgiev, A. Sillitti, and G. Succi, "A study of energy-aware implementation techniques: Redistribution of computational jobs in mobile apps," *Sustainable Computing: Informatics and Systems*, vol. 7, pp. 11–23, 2015.

[11] B. Patra, S. Roy, and C. Chowdhury, "A framework for energy efficient and flexible offloading scheme for handheld devices," in *Advanced Networks and Telecommuncations Systems (ANTS), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1–6.

[12] S. A. Saab, A. Chehab, and A. Kayssi, "Energy efficiency in mobile cloud computing: Total offloading selectively works. does selective offloading totally work?" in *Energy Aware Computing Systems and Applications (ICEAC), 2013 4th Annual International Conference on*. IEEE, 2013, pp. 164–168.

[13] E. Benkhelifa, T. Welsh, L. Tawalbeh, Y. Jararweh, and A. Basalamah, "Energy optimisation for mobile device power consumption: A survey and a unified view of modelling for a comprehensive network simulation," *Mobile Networks and Applications*, vol. 21, no. 4, pp. 575–588, Aug 2016.

[14] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: A measurement study and implications for network applications," in *Proc. of the 9th ACM SIGCOMM Conf. on Internet Measurement*, ser. IMC '09, 2009, pp. 280–293.

[15] F. Mehmeti and T. Spyropoulos, "Performance modeling, analysis, and optimization of delayed mobile data offloading for mobile users," *IEEE/ACM Trans. Netw.*, vol. 25, no. 1, pp. 550–564, Feb. 2017.

[16] M. McMillan, *Data Structures and Algorithms Using C*. Cambridge University Press, 2007.

[17] L. M. Fischer, L. B. de Brisolara, and J. C. B. de Mattos, "Sema: An approach based on internal measurement to evaluate energy efficiency of android applications," in *Computing Systems Engineering (SBESC), 2015 Brazilian Symposium on*. IEEE, 2015, pp. 48–53.

[18] A. C. Harper and R. V. Buress, *Mobile Telephones: Networks, Applications, and Performance*. Nova Publishers, 2008.

[19] T. Salah, M. J. Zemerly, C. Y. Yeun, M. Al-Qutayri, and Y. Al-Hammadi, "The evolution of distributed systems towards microservices architecture," in *Internet Technology and Secured Transactions (ICITST), 2016 11th International Conference for*. IEEE, 2016, pp. 318–325.