Aves: A Framework for Energy-efficient Stream Analytics across Low-power Devices

Roshan Bharath Das, Marc X. Makkes, Alexandru Uta, Lin Wang, Henri Bal
Vrije Universiteit Amsterdam
{r.bharathdas, m.x.makkes, a.uta, lin.wang, h.e.bal}@vu.nl

***Abstract*—Today's low-power devices, such as smartphones and wearables, form a very heterogeneous ecosystem. Applications in such a system typically follow a reactive pattern based on stream analytics, i.e., sensing, processing, and actuating. Despite the simplicity of this pattern, developing applications has become increasingly difficult, especially when a large number of devices with different platforms are involved. In addition, deciding where to place the processing tasks of an application to achieve energy efficiency is non-trivial in such a heterogeneous system since application components are distributed across multiple devices.**

In this paper, we present Aves – a framework that provides a set of unified APIs for programming applications in such distributed environments. Aves also incorporates a decision-making engine based on a holistic energy-prediction model, with which the processing tasks of applications can be placed automatically in an energy-efficient manner without programmer/user intervention. We validate the effectiveness of the model and reveal several counter-intuitive placement decisions. Our framework's improvements are typically 10-30%, with up to a factor 14 in the most extreme cases. We also show that Aves's decision engine gives an accurate decision in comparison with real energy measurements for two sensor-based applications.

I. Introduction

With the introduction of Internet-of-Things and 5G, smart sensors will become ubiquitous and will generate massive amounts of data. These sensor-based Big Data are of increasing importance in many fields, such as smart homes, cities, or farming [1], [2], [3]. Such sensors are building blocks of an extremely heterogeneous ecosystem: different devices have different processing power, battery capacity, networking capabilities, and programming environments. Therefore, it is essential to provide a programming environment with which efficient sensor-based applications can be conveniently built.

A typical example of such use cases is building healthcare applications [4], [5]. In this case, a smartwatch measures the patient's heart rate. A smartphone can then analyze the gathered data and identify patterns in the patient's heart rate. However, if the data analysis is too complex to be performed on a smartphone, the computation could be offloaded to a nearby cloudlet or a remote cloud. The analysis is usually followed by a decision, and an actuation is performed accordingly (e.g., a message is sent to either the patient or the doctor). Developing such an application is intrinsically complex: the programmer needs to reconcile different APIs specific to different platforms, but also complex decision-making mechanisms have to be developed for efficient placement of distributed subsystems.

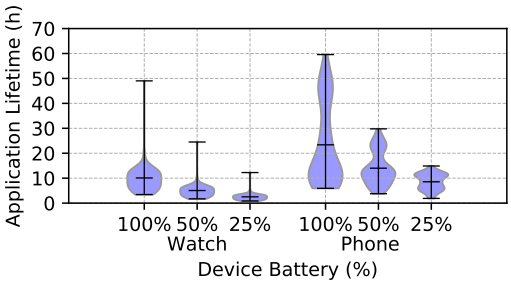


Fig. 1: Variation of application lifetime for different types of processing task for a device with a given initial battery level. This variation is further exacerbated by the available amount of battery and the type of device (smartphone or smartwatch).

To reduce development complexity and aid processing task offloading decisions, we advocate a framework with a unified programming model for stream analytics on low power devices. Such a framework helps programmers by providing a set of unified APIs that are easy to use, and allow for automatic processing task placement decisions based on device capabilities and application processing complexity. We believe that a *sensing-processing-actuating* model is sufficient to build the majority of sensor-based applications that are based on stream analytics. We denote the different steps in the sensing-processing-actuating model as sensing-task, processing-task and actuation-task.

One crucial deciding factor for processing-task placement is the maximum time the application can run with the remaining battery capacities of the involved devices. Nowadays, for battery-powered devices, such as smartphones or wearables, it is crucial to optimize battery usage. A typical solution to prolong battery life is to offload complex processing or actuation tasks to cloudlet or cloud platforms [6]. While in such one-to-one offloading scenarios optimizing the application lifetime can be simply mapped to maximizing the energy efficiency of single devices, this is not sufficient when we are facing a distributed heterogeneous environment, where multiple devices are present with different properties, e.g., battery capacities, processing power, memory, communication technologies (e.g., Bluetooth, WiFi, 4G, Ethernet), and software.

Deciding where to place processing-tasks for maximizing the lifetime of an application is challenging, especially with the introduction of both local and remote sensing and actuation in a distributed environment. One reason is that the offloading decisions are typically difficult when multiple heterogeneous

devices are involved, as depicted in Figure 1. The measurements are from a synthetic benchmark set. The difficulty is due to the variation in the application lifetime for different combinations of parameters such as the complexity of processing, the frequency of sensor data generation, the processing window size, and the battery charge of the devices involved. In addition, the fact that data needs to be moved along with the processing-task incurs communication overhead that also consumes considerable energy, which further complicates the problem. The location of sensing- and actuation-tasks plays a significant role in the offloading decision making. All these render manual decision making impractical.

In this paper, we introduce Aves, a programming framework for energy-efficient stream analytics across low-power devices and clouds. Aves enables a unified programming environment for building sensor-based applications that can perform sensing-, processing-, and actuation-tasks in heterogeneous environments. Aves is also equipped with an automatic decision-making engine for processing-task placement, based on efficiently exploring the large space of offloading policies and employing a holistic energy model to estimate how long a given application is able to run under a given policy. The energy model uses energy consumption measurements from a collection of synthetic benchmarks that are run once ('offline') for the given platform (smartphone, smartwatch). When an application is launched, Aves automatically decides where to do the processing, based on the application-specific computation complexity, the window size, the sensor frequency, and the current battery levels of the phone and watch. Here, Aves uses estimation among the pre-measured synthetic values, in combination with an energy model.

Our main contributions are summarized as follows:

- We present a unified programming framework for developing sensor-based applications that are based on stream analytics on heterogeneous devices, reducing the application development complexity (Section III).
- We propose and incorporate an automatic decision-making engine to the framework, which can place processing-tasks for applications that are based on stream analytics in an energy-efficient manner using a holistic energy model (Section IV).
- We validate our energy model and decision-making engine with real-world scenarios, confirming the effectiveness of the framework (Section V).

II. Background and Motivation

In this section, we describe various sensor-based applications that can be built using the Aves framework. Then, we discuss stream analytics patterns and the possibility of processing-task offloading. Further, various metrics for offloading decisions are described.

A. Applications

Context-aware applications can be built using various sensors from devices such as smartphones, smartwatches, or other IoT devices. The data collected from the sensor is processed, and some action is performed based on the result. There can be various applications that involve local and remote sensing and actuation. We provide five concrete scenarios that use sensors from a smartwatch and perform actuation remotely on a smartphone, or a cloud or cloudlet. In this paper, we will use cloudlet to denote either a remote cloud or a local cloudlet.

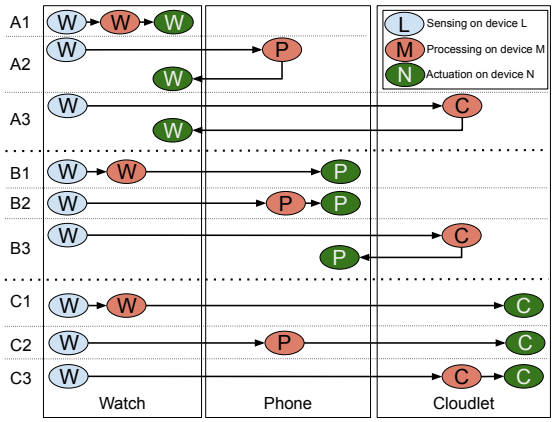


Fig. 2: Various stream analytics patterns in applications that are built across multiple heterogeneous devices with sensing-task on a smartwatch.

These scenarios require sensing on the smartwatch and remote actuation on the phone:

Elderly care app: In this application an elderly person's average heart rate needs to be continuously monitored to identify any abnormal behaviors and the caretaker should be notified when help is necessary. The heart rate sensing occurs on the elderly person's smartwatch and the actuation (notification) occurs remotely on the care taker's smartphone.

Coaching app: This application in the context of rowing gathers sensor data from the rower's watch and the coach is notified about the rower's statistics. In such a scenario, the movement (accelerometer) sensing occurs on the rower's smartwatch and the statistics update occurs remotely on the coach's smartphone.

These scenarios require sensing on the smartwatch and remote actuation on the cloudlet:

Environmental monitoring app: This application can gather a median of light sensor data along with GPS from multiple user's watches to spot dark areas in a city for safety purposes.

Noise detection app: It can gather sound sensor data from multiple watches in an area and perform Fast Fourier Transform (FFT) to identify noise pollution (such as airplane, explosion, etc).

Sleep pattern app: It concerns identifying a person's sleep pattern and co-relating it with some other data sets such as disease-related data that require processing over large data sets in the cloud.

Next, we discuss various stream analytics patterns in these applications and the possibility of offloading processing-tasks for some scenarios.

B. Stream Analytics Patterns

Figure 2 illustrates various cases in stream analytics for applications that are built across the smartwatch, the smartphone, and the cloudlet, as already covered in the above scenarios. When sensing is fixed on the smartwatch, we have 9 different cases. Similarly, we can build other cases with sensing on the smartphone or on the cloudlet.

For a sensing-processing-actuating scenario LMN, L represents sensing on the device L, M represents processing on the device M and N represents actuation on the device N. This notation will be used throughout the paper. Note that the devices L, M and N can be a watch, phone or cloudlet. Therefore, a scenario WPC implies that the sensing occurs on the watch (W), processing occurs on the phone (P) and actuation occurs on the cloudlet (C). When the processing-task could be on either phone, watch or cloudlet, we keep the notation 'X'. For example, for the scenario WXC, the processing occurs in a device X which could be either a watch, phone or cloudlet.

From figure 2, we observe three main patterns that include sensing and actuation tasks. Case A1 represents the first pattern (WWW) where both sensing and actuation occur locally. Case B1 (WWP) and C1 (WWC) are when sensing occurs locally on the smartwatch and actuation occurs remotely on the smartphone and the cloudlet, respectively. For these cases, we can offload the processing-task. For example, for A1, when we offload the processing-task to the smartphone, it becomes case A2 (WPW). Similarly, case A3 (WCW) is for offloading the processing-task to the cloudlet. We note that for a given scenario, there exist multiple options to offload the processing-task. Hence, it is important to know the most energy-efficient way to do the processing given a scenario at hand.

C. Metrics for Offloading Decisions

On offloading the processing-task, we can improve various metrics such as latency, throughput, and energy. The latency can be improved by performing simple processing locally on the device and offloading complex processing to remote devices that are nearby and have better capabilities in terms of computing resources. The throughput is another metric for offloading the processing-task. In case of stream analytics, when the processing frequency increases, the number of processing operations that need to be done also increases and it may reach the maximum sustainable throughput especially when data from multiple sensors needs to be processed. Energy is another important metric especially with battery-powered devices such as smartwatches and smartphones, it becomes important to offload processing to save energy. Especially with remote actuation, there is a possibility to do processing at the sensing device (WWP), at the actuation device (WPP) or a different device (WCP). In particular, we focus on a new metric related to energy efficiency, i.e., **maximizing the application lifetime**, by deciding where to place the processing-task given a scenario at hand. Next, we see how our framework can be used for this purpose.

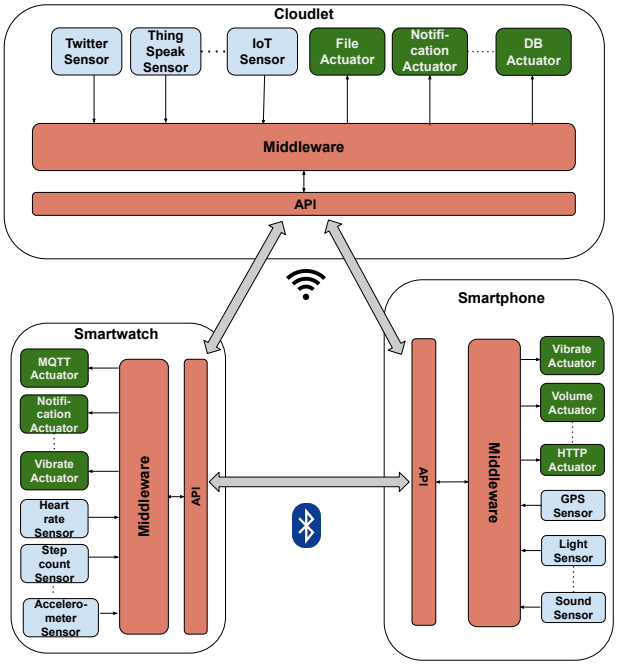


Fig. 3: An overview of the Aves architecture. (Sensors are light blue, actuators are dark green.)

III. Aves Framework

In this section, we discuss the architecture and the programming abstraction of the Aves framework. It aims at helping developers to easily build energy-efficient context-aware applications in a heterogeneous environment. We focus on improving the energy efficiency for small footprint edge devices (such as smartphones, Rasberry Pis, smartwatches) by providing simple (less expressive) functionality at the edge. Our work in conjunction with centralized analytic systems (such as Storm [7] and Spark [8]) that support more complex analytics, can be used to provide energy-efficient analytics across the whole IoT ecosystem.

A. Architecture

Figure 3 shows the architecture of the Aves framework. It contains three different types of nodes, the watch, the phone, and the cloudlet. The watch and the phone interact with each other using Bluetooth communication. Both the watch and the phone interact with the cloudlet using WiFi or 4G communication. In a heterogeneous environment, every node can perform sensing, processing, actuation, or a combination of it. We note that the watch, the phone, and the cloudlet can perform all three functions: sensing, processing, and actuation. Also, multiple watches and phones can interact with each other via the cloudlet. Currently, the framework supports both Android and Java-based IoT devices.

Next, we describe the components of the Aves framework.

1) Sensors: Sensors generate continuous data that needs to be processed. Sensors can be either hardware (e.g., GPS, accelerometer, gyroscope) or software (e.g., weather, twitter, news, stock). In the case of software sensors, the data may be

generated locally or gathered remotely. Different types of sensors generate data at different frequencies. Some sensors also allow different frequencies. For example, an accelerometer sensor can generate data at four different levels of frequencies where the fastest is used for gaming purposes and the slowest for screen orientation changes.

2) Actuators: The sensor data is evaluated, and the result is sent to the application for further action. The actuation can be hardware-based such as vibrate, turn on the flashlight or software-based (e.g., send notifications, log, make HTTP requests). For sending the evaluation result from the phone to a server, actuators such as HTTP or MQTT [1] are used. The result received at the server as a part of actuation from the phone can be used as sensor data for further processing. In this way, we can enable easy data flow between nodes.

3) Middleware: The middleware is responsible for four main tasks: 1) gathering relevant sensor data from various sensors 2) processing the gathered sensor data based on a given window size, frequency and operation 3) performing both local and remote actuation 4) inter-device communication such as Bluetooth communication between the watch and the phone and WiFi communication between watch or phone to the cloudlet.

4) API: The Aves API handles incoming requests from other devices or third-party applications. Using this API, the developers can easily subscribe to sensor data, trigger actions, or get the list of sensors and actuators available on a device.

To implement Aves, we extend our Swan [9] framework. The Swan framework already runs on resource-constrained devices, and provides support for distributed sensing, offloads computation from phone to cloud, performs stream analytics on the cloudlet and makes use of an easy to use domain specific language called Swan-song [10]. However, the Swan middleware does not provide interaction between heterogeneous resource-constrained devices (phone and watch), and there is no support for actuation (both local and remote), which are vital functionalities to build applications that follow the sensing-processing-actuating model.

To this end, we implement sensors, actuators, middleware, and API for the watch on top of the Swan framework. In particular, we introduce distributed actuation by enabling interaction such as register or unregister expressions and send or receive sensor data between the phone, the watch, and the cloudlet. We also extend the domain specific language used by the framework to enable distributed actuation (see Section III-B). Using this addition, we enable the easy building of context-aware applications that involve local and remote sensing, processing, and actuation on a combination of heterogeneous devices such as watches, phones, and cloudlets.

B. Programming Abstraction

The Aves framework allows application developers to easily interact with various sensors and actuators. The developers can easily register expressions written using an extension of Swan-song to perform actuation locally and remotely.

[1] https://mqtt.org/

Out of all possible types of scenarios, the paper focuses on two representative scenarios that include local and remote sensing, processing, and actuation. The first scenario LLM performs the sensing and processing on the $deviceL$ and the actuation on the $deviceM$ and the expression for this scenario is written as:

```
deviceL@sensor:path{operationX,timeWindowY}
THEN
deviceM@actuate:path
```

where the sensing expression and the actuation expression are separated by $THEN$, $deviceL$ represents the location of sensing and $deviceM$ represents the location of actuation, $sensor : path$ implies the type of the sensor and the value path (e.g., location:latitude), $operationX$ represents the type of operation (e.g., MEAN, MAX) and $timeWindowY$ represents the time window (e.g., 10s for 10 seconds), $actuate : path$ represents the type of the actuation and the value path (e.g., vibrator:vibrate). The expression computes the operation over a time window on the data generated by the sensor on the $deviceL$, and the result is sent to the $deviceM$ for actuation.

The second scenario is LMN where the first expression only gathers data from the sensor on the $deviceL$ and sends it to the $deviceM$ without doing processing (also called as ANY processing) and in the second expression the $deviceL$ gathers the actuated data as a sensor and performs processing. The result is sent to the $deviceM$ for actuation. The expressions are written as:

```
/* Expression registered on deviceL */
deviceL@sensor:path{ANY,0s}
THEN
deviceM@sensorA:pathE
-----------------------------
/* Expression registered on deviceM */
deviceM@sensorA:pathE{operationX,timeWindowY}
THEN
deviceN@actuate:path
```

IV. Energy Model and Decision Making

As discussed, the main goal for processing-task placement in Aves is to maximize the application lifetime. To this end, we built an energy model to enable the decision-making process, which involves finding the impact of various stream analytics parameters such as frequency, window size, and operation complexity on the energy measurement. For a given scenario, the energy is measured for a possible combination of parameters, and then curve fitting is applied on the training set. The trained model is used to predict the energy measurement for a given set of parameters. Then, the decision on where to do the processing is made based on the predicted energy measurement and current battery level for each device. The decision engine registers the relevant expression accordingly.

A. Applicability

The application developer registers an expression using the Aves API on the phone. From the expression, the location

and the type of sensors and actuators are inferred along with the type of operation and the window size. Aves has already classified the operations based on their complexity and the sensors based on their frequency, and it can also identify any additional configuration (sensor delay) set by the developer. An expression to measure the average heart rate over 10 seconds from the watch and to perform a vibration actuation on the phone can be written by the developer as:

watch@heartrate: value{MEAN,10s}
THEN
phone@vibrator: vibrate ? duration =500ms

where $watch$ represents the location of sensing and $phone$ represents the location of actuation, $heartrate : value$ implies that it is a heart rate sensor, $MEAN$ represents aggregation operation and $10s$ represents the time window, $vibrate : vibrate$ represents that it is a vibrate actuation and $duration = 500ms$ represents the duration of vibration which is 500 milliseconds in this case.

After inferring the expression, Aves checks all possible options for placing the operation and detects the best possible scenario for the given expression. The best choice is identified based on an energy model and a decision engine. Using the energy model, the electric current for a given scenario with given parameters is estimated. The decision engine further uses the estimated electric current for all possible scenarios to identify the best possible choice with maximum application lifetime based on the present battery level of the devices that are involved. In the above scenario, the sensing occurs on the watch and the actuation occurs on the phone. Assuming that there is a nearby cloudlet, there is a choice to do the processing $(MEAN, 10s)$ on the watch, on the phone or the cloudlet. The decision engine chooses the scenario that can maximize the application lifetime to do the processing based on the battery levels of the watch and of the phone and registers the relevant expression. In this case, we assume that the battery level of the cloudlet can be ignored as it is connected to the power supply. The decision on where to do the processing is complex as it changes based on multiple factors such as battery level, sensing frequency, operational complexity, window size and the location of sensing and actuation. Hence, Aves helps the developers to ease the application development process.

B. Evaluation Setup

We chose electric current instead of power to model energy for two reasons. First, the change in the voltage for both smartphones and smartwatches are minimal. Therefore, power (voltage×current) has an impact only on the electric current. Second, to measure the battery life for performing decision making, electric current is used instead of power (as shown in equation 4).

The electric current can be measured based on a hardware-based power monitoring tool such as Monsoon [11] or software-based such as Trepn profiler [12]. Trepn is more widely available, but the measurement from Monsoon is more accurate. For our experiments, we use Monsoon. It was

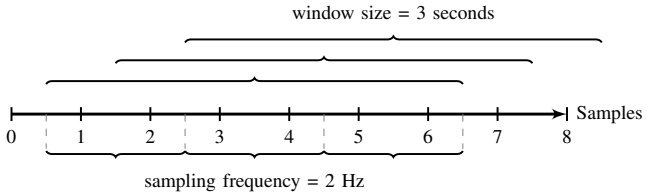


Fig. 4: Event processing scenario.

attached to the watch and the phone to measure the electric current usage. We measure the average electric current over 5 seconds (30,000 data points) for various scenarios. The voltage was set to 4 V for all the experiments. As a watch, we use the Moto 360 2nd gen with a battery capacity of 375 mAh, and for phone, we use Nexus 5x with battery capacity 2700 mAh as shown in Table I. Both the watch and the phone communicate with each other using Bluetooth and with the cloudlet using WiFi communication.

C. Parameters

Figure 4 shows an event processing scenario that operates on a window size of 3 seconds and the sensor data collection rate of 2 Hz. On every new event, the operation is performed over the window size of 3 seconds. Since the frequency is 2 Hz, the window size will contain six samples. For the MEAN operation with complexity $\mathcal{O}(N)$ the average over six samples is calculated. Also, other operations with different complexities can be used depending on the scenario. Hence, the number of operations (ϕ) depends on the frequency (f), the window size (w) and the complexity of the processing ($\mathcal{O}$) and it can be shown as:

$$\phi = \mathcal{O}(g(n)) \tag{1}$$

where $n = f \cdot w$ and $g(n)$ is a function that represents the type of processing such as MEAN, MEDIAN that can have different time complexities.

D. Synthetic Workload

In general, it is not doable to do measurements and modeling for each new application. Our solution is to measure (once, offline) a broad spectrum of synthetic cases to build the energy model. Applications with a given set of parameters (complexity, frequency, window size) are estimated based on the model and the best option (e.g., WWP) that can maximize the application lifetime is chosen. In our previous work, we have already used an online profiler-based approach [13] to improve the energy-efficiency. However, software profiling will take some time in the beginning to measure the electric current. With the offline model-based approach we can make fast decisions in the beginning and in combination with our earlier work we can also validate or change the decisions in the run-time.

Figure 5 shows the battery life of two scenarios (WXP and WXC) for various possible combinations of parameters. WXP indicates sensing on the watch and actuation on the phone

TABLE I: Device configuration.

Resource type	Device	Operating system	CPU	Memory	Network Communication	Battery Capacity (mAh)
Watch	Moto 360 2nd gen	Wear OS 2.0	1.2 GHz quad-core CPU (only one core enabled)	512 MB	Bluetooth 4.0 WiFi 802.11 b/g	375
Phone	LG Nexus 5X	Android 8.1.0	Hexa-core (4x1.4 GHz Cortex-A53 & 2x1.8 GHz Cortex-A57)	2 GB	Bluetooth 4.2 WiFi 802.11 a/b/g/n/ac	2700
Cloudlet	MacBook Pro	macOS Sierra 10.12.6	8 core (2.8 GHz Intel Core i7)	16 GB	WiFi 802.11a/b/g/n	N/A (connected to power supply)

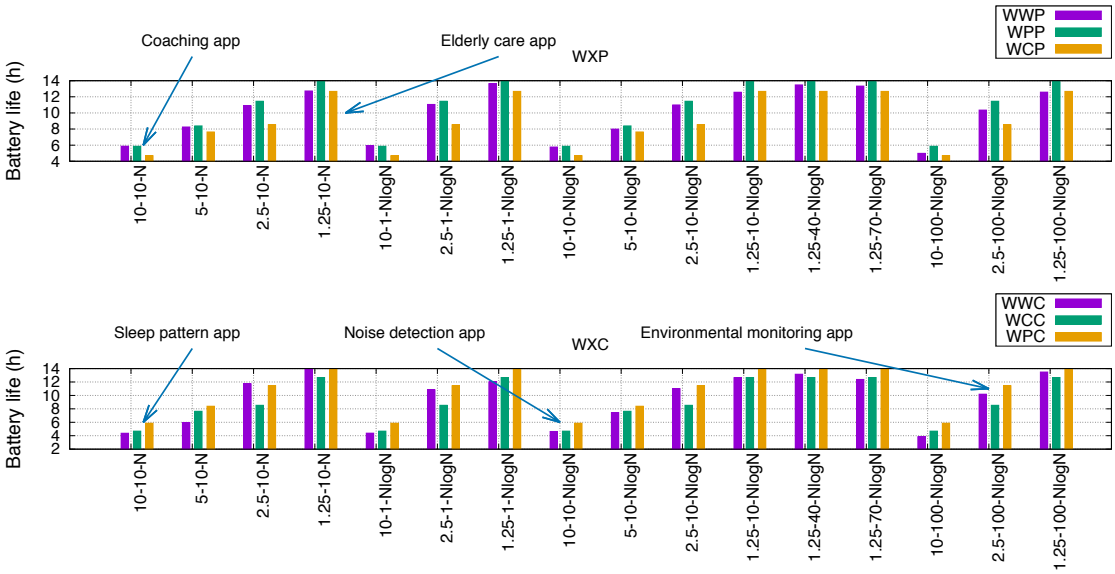


Fig. 5: Battery life comparison of three choices for two cases (WXP and WXC) using various synthetic workloads.

and WXC indicates sensing on the watch and actuation on the cloudlet. The battery life in the figure represents the total time the application can run in hours when both the phone and the watch are at the battery level of 100%. On the x-axis, the label $10-10-N$ indicates a combination of sensor frequency, window size, and operational complexity, respectively. From the figure, in case of WXP and for $10-10-N$, there are three possibilities to perform processing: WWP (locally on the watch), WPP (remotely on the phone) and WCP (remotely on the cloudlet). WCP is the least preferred choice for that combination of parameters. We also note that for a different combination of parameters the best choice also changes.

The figures show five applications that were discussed in Section II-A: coaching, elderly care, sleep pattern, noise detection, and environmental monitoring. Such applications use different combinations of sensors, actuators, and parameters.

E. Energy Model

The energy model is built based on the two basic blocks: the computation on the device and the communication between the devices. The computation is represented in terms of the number of operations (ϕ) associated with a given analytics scenario and the communication is represented in terms of frequency of remote actuation (f). Here, we show the impact of computation and communication on the electric current for both the watch and the phone.

Figure 6 shows the average electric current on varying frequency and window size for both the watch and the phone. The figure is used to identify the relationship of various parameters (independent variables) to electric current (dependent variable). The average electric current is measured separately for each device that is involved in a scenario. To show the impact of frequency, out of all possible scenarios, we show the interesting scenarios such as local sensing, processing and actuation (WWW for watch, PPP for phone), local sensing and processing with remote actuation (WWP and WWC for watch, PPW and PPC for phone), remote sensing with local processing and actuation (PWW for watch, WPP for phone) and remote sensing and processing with local actuation (PCW for watch, WCP for phone). For measuring one parameter (e.g., frequency), we keep other parameters (window size and complexity) constant. To show the impact of window size we chose the scenario that involves only computation, i.e., local sensing, processing and actuation (WWW for watch, PPP for phone). We note that for various scenarios, the average electric current is different. The variation implies that offloading the processing-task will have an impact on the average electric current. For a given scenario, we see a linear increase in the electric current used when both the frequency and the window size increase.

Based on the observations, we see a linear correction between various parameters and the electric current. In contrast, a non-linear model would have made it very difficult to interpret and use. Since we have a linear correlation, we can use a simple model using multiple linear regression analysis

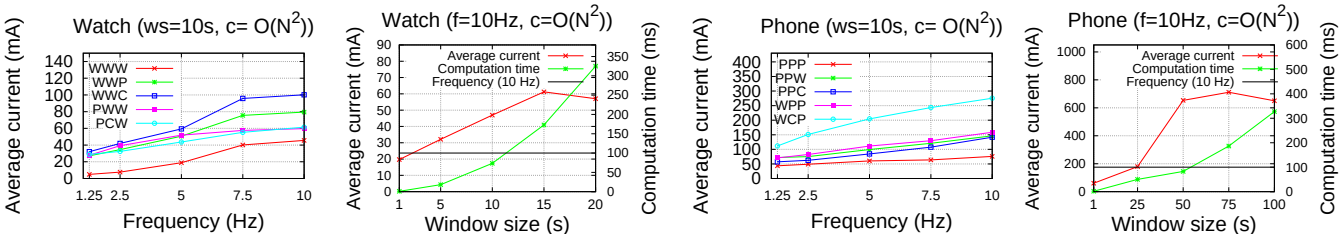


Fig. 6: The impact of frequency and window size on the electric current measurement for multiple scenarios on both watch and phone. The symbol f, ws and c represents frequency, window size and complexity respectively.

to model the electric current. For choosing the variables, we note that the electric current is mainly impacted by the sensing frequency, the processing frequency, the amount of processing and the actuation frequency. For expressions where the frequency for sensing, processing, and actuation are the same, we chose the frequency (f) as one variable. The other variable is the number of operations (ϕ) which indicates the amount of processing, and it is chosen based on the frequency, window size, and the complexity as seen in equation (1).

It is important to note that various hardware sensors and actuators will have varying energy usage. However, across multiple offloading possibilities, energy consumption will remain the same. For example, for a scenario WXP that uses accelerometer sensor on the watch and vibrate actuator on the phone, the operation offloading possibilities (WWP, WPP, and WCP) will consume the same amount of energy for accelerometer sensor and vibrate actuator. Hence, the energy consumed by the sensor will not have an impact on the energy model. Therefore, we measure the electric current using a test sensor and actuator for the synthetic cases. For making the model simple, we assume that the network transmission error is minimal. The model for measuring the electric current (δ) can be written as:

$$\delta = \beta_0 + \beta_1 f + \beta_2 \phi \tag{2}$$

For expressions where the frequency of sensing (f_1) and actuation (f_2) are different, the model will then be written as:

$$\delta = \beta_0 + \beta_1 f_1 + \beta_2 f_2 + \beta_3 \phi \tag{3}$$

From the model, the regression coefficients (β_n) are estimated based on several samples for each scenario. The estimation is done using curve fitting based on the least square method. It aims to minimize the difference between observed and predicted values.

F. Decision Making

While our decision engine decides where we should perform computation, we also have to take into account the remaining battery charge, as it can influence the decision where to process the sensor data. Figure 7 shows the influence of the remaining battery charge on the discharge rate of both watch and phone. Here, we fix the battery level of the watch and vary the initial battery charge of the phone. In addition, the figure

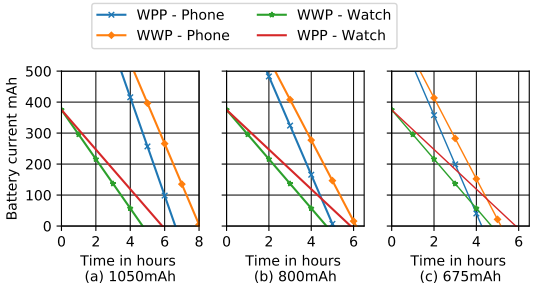


Fig. 7: Decision making based on remaining battery life.

shows two scenarios, where sensing and actuation are fixed. The first scenario shows two lines, i.e., sensing on the watch in combination with processing and actuation on the phone (WPP). The second scenario shows the lines for both sensing and processing on the watch and only actuation on the phone (WWP). For the watch, the average electric current on running the program while processing the data locally is 79.45 mA, not processing but offloading to the phone the electric current of the watch is 63.98 mA. In case of the phone, the electric current for processing on the watch is 130.69 mA (for using Bluetooth communication), and for processing locally on the phone, it is 158.61 mA. The overall goal of the decision engine is to run the application longest using these two devices.

Figure 7(a) shows the discharge rate of WPP and WWP for the phone exceeds the watch. The maximum lifetime is determined by the highest discharge rate of the devices involved for a given scenario, which in the case of WPP is WPP for the watch at 5.9 hours, and not WPP for the phone. Figure 7(b) shows the discharge rate of WPP for the phone exceeds WPP for the watch. However, we still achieve slightly longer lifetime using processing-task on the phone (WPP). On the other hand, Figure 7(c) shows that keeping the processing-task locally in the watch (WWP) will yield the longest application lifetime. Trivially, and not shown in Figure 7, if the lifetime of both WWP and WPP for the phone is lower than WWP and WPP for the watch, the application lifetime will be at most the lowest discharge rate of the phone, i.e., offloading processing-task to the watch (WWP). Hence, to span the lifetime of a distributed application, we have to take the remaining battery charge into account.

Now, we can generalize the decision-making process to more than two devices (namely phone and watch). A single

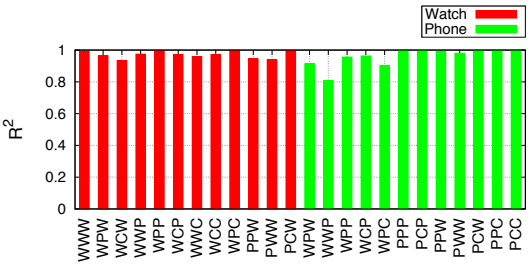


Fig. 8: R-squared value for various scenarios.

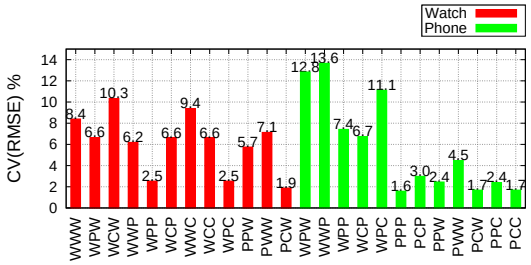


Fig. 9: CV(RMSE) for various scenarios.

scenario L_i is computed as follows:

$$L_i = \min\left(\frac{P_{B_1} \cdot C_{B_1}}{\delta_{B_{1_i}}}, \frac{P_{B_2} \cdot C_{B_2}}{\delta_{B_{2_i}}}, ..., \frac{P_{B_m} \cdot C_{B_m}}{\delta_{B_{m_i}}}\right), \quad (4)$$

where $B_{1...m} \in \{Phone, Watch, ...\}$ (battery-powered devices), C_{B_m} is the battery capacity for the device B_m, P_{B_m} represents the present battery percentage for the device B_m, and $\delta_{B_{m_i}}$ (gathered from the equations 2 or 3) is the electric current for a scenario i for the device B_m. Equation 4 gives the result of the device with the shortest battery life for a given scenario. By applying the following equation:

$$\alpha = \max(L_1, L_2, ..., L_n). \quad (5)$$

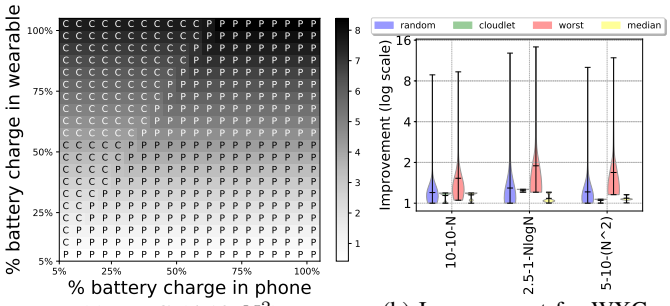
We will find the scenario α with the longest application lifetime out of the possible n scenarios. Note that the devices in the equation 4 are battery-powered. It is possible that for some devices, battery consumption is not an issue. For example, a laptop that is connected to the power supply may not necessarily be concerned about battery life. In such cases, we ignore the measurement of battery life for those devices in equation 4.

V. Evaluation

In this section, we give a brief summary of the various evaluations performed. We describe how the validation of the energy model is performed. Then, we show the impact of the battery percentage on the offloading decision and compare it with various offloading strategies. Finally, we compare the decision made by Aves with a hardware power monitor for two real-world applications.

A. Summary

We describe various evaluations performed using the framework and address the following questions: 1) What kind of model do we need? Is a linear model a good fit? 2) How accurately can the model predict the electric current measurements? 3) What is the impact of battery life and battery percentage of various devices on the offloading decision? 4) How much is the maximum improvement of the application lifetime for a given scenario? Does offloading to the cloudlet save energy for any scenario? 5) How accurately can Aves estimate the best option for real-world applications?



(a) WXC (5-10-N^2). (b) Improvement for WXC.

Fig. 10: Impact of battery percentage on the offloading decision.

B. Validation of the Model

We use 23 samples per scenario out of which 80% of the samples are used for training the model and 20% for prediction. We measure two aspects: coefficient of determination (R^2) [14] and the coefficient of variation of the root mean square error ($CV(RMSE)$). The R^2 value indicates the goodness of fit and is measured between 0 and 1. Figure 8 shows the R^2 value for various scenarios. We note that the R^2 value for 23 out of 24 scenarios is greater than 0.90 out of which the value for 18 scenarios is greater than 0.95. Only 1 scenario (WWP) has R^2 value of 0.80. The result implies that the linear regression model is a good fit.

Next, we measure the variability of errors between the observed values and the predicted values for each scenario using $CV(RMSE)$. It is measured in percentage, and a lower value indicates a higher accuracy for the model. Figure 9 shows the $CV(RMSE)$ for various scenarios. In the figure, we note that the $CV(RMSE)$ percentage is less than 10% for 20 scenarios out of which the value is less than 5% for 10 scenarios. For only 4 scenarios, the value is between 10% and 15% with the highest value for WWP (13.6%). The result indicates that the model is highly accurate.

C. Offloading Decision

Figure 10 shows the impact of battery percentage on the offloading decision. We choose the scenario: WXC. WXC does sensing on the watch and actuation on the cloudlet. Figure 10a shows the heat map of the chosen device to offload processing for a given battery percentage of the phone and the watch. The letter 'P','C' indicates that the processing on the phone, and the cloudlet respectively is the best decision. The intensity represents the application lifetime in hours. The parameter $5-$

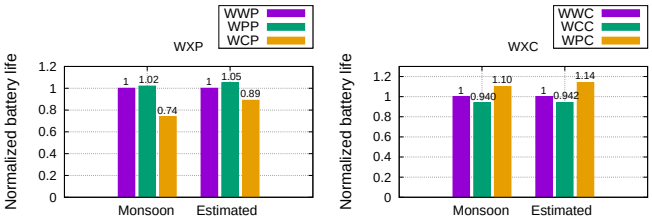


Fig. 11: Normalized battery life comparison of Monsoon hardware power monitor with the decision-engine's estimation for two real-world applications. The battery percentage for both phone and watch are at 100%.

$10 - N^2$ indicates a frequency of 5 Hz, a window size of 10 seconds and a complexity of $\mathcal{O}(N^2)$.

For scenario WXC, when the battery percentage of watch and phone is more than 75% and less than 50% respectively, it is better to do processing on the cloudlet. On the other hand, when the battery percentage for the watch is below 50% and the phone is above 50%, it is better to do processing on the phone. Hence, it is better to offload the processing-task to the phone or the cloudlet than processing locally on the watch. Note that this result is specific to a given scenario with given window size, frequency, and complexity.

Figure 10b shows the improvement (in number of times) for three different parameters. We compare the decision engine's choice of task placement with the random, the cloudlet-based, the worst, and the median-based choices. In the plot, the x-axis represents parameters with different frequency, window size, and complexity, the y-axis represents the improvement for different combinations (from 5% to 100%) of battery percentage of both the phone and the watch, the width shows the probability density of the data at different values and the horizontal stripes show the minimum, the mean and the maximum value. We observe a maximum improvement of 14.25 times. With the parameter $2.5 - 1 - NlogN$, we notice an average improvement of 1.33, 1.31, 1.91 and 1.11 times. Hence, Aves always gives good decisions, whereas random or manual choices may suffer from much more variation.

D. Real Application Comparisons

As described in Section IV-A, Aves makes automatic placement decisions for applications without doing new measurements (instead, it uses estimation based on pre-measured synthetic benchmarks). Here, we evaluate how well this automatic decision-making works. We study two applications, and compare the decisions made by Aves (through estimation) against the best decision (determined by measuring). The first application is an elderly care app that is used to measure the average heart rate over 10 seconds of an elderly person and displays it on the phone of the caretaker. This application follows a WXP scenario with parameters $1-10-N$. Since the benchmark set only contained measurements for the closest set of parameter $1.25-10-N$, the measurements for $1-10-N$ had to be estimated using equation 2.

The second application does simple environmental monitoring using $\mathcal{O}(N)$ analysis. The app measures the average light sensor data over 30 seconds from multiple smartwatches and sends it to the cloudlet for further analysis. This application follows a WXC scenario with parameter $3-30-N$. In this case, the benchmark set only contained measurements for the closest set of parameter $2.5-10-N$. Hence, the measurement had to be estimated using equation 1 and 2 for each scenario.

To evaluate our automatic decision engine, we have also actually measured the electric current for all combinations of the applications, using the Monsoon hardware power monitor. It is then compared against the estimated electric current for each scenario. Figure 11 shows the results of the comparison. The battery life measurement is normalized against the local processing (WWP and WWC) to remove the additional current usage caused by sensors and actuators. Removing the overhead will not have an impact on the decision-making process, as described in Section IV-E.

In all cases, the decision engine and the measurement based on the Monsoon power monitor chose the same, i.e., WPP for WXP, and WPC for WXC. In the two cases, the maximum application lifetime is attained by using the remote device (phone). Hence, Aves helps the developers to build energy-efficient sensor-based applications easily.

VI. Related Work

Below, we compare Aves against other frameworks for sensing/actuation and offloading.

Sensing and Actuating Frameworks - Various solutions exist that can help in building context-aware applications. IFTTT [15] uses trigger action-programming to build context-based recipes. They do not focus on building recipes based on sensor data. Tasker [16] allows adding rules through a graphical interface. However, they are not suitable for use as middleware for building complex sensor-based applications.

There are several stream processing platforms [8], [7] built for distributed processing on large clusters. They focus on processing data collected from smartphones and wearables in the cloud. However, they are not suitable for some scenarios that require processing locally to save energy. Alternatively, Edgent [17] is built to run on low-end devices. However, there is no domain-specific language to build applications easily. Some frameworks have been proposed to build context-based applications. Sentio [18] focus on virtualization of sensors and runs as middleware on the watch, the phone, and the cloud. They do not provide programming support for offloading the processing in a heterogeneous environment.

Our solution is different in that we enable programmers to build energy-efficient context-aware applications in a heterogeneous environment that contains different types of nodes.

Energy-efficient Computation Offloading - Computation offloading has been intensively studied [19]. Its goal is typically reducing the execution time of smartphone-based applications or improving the energy efficiency of mobile devices. In most cases, computation from a phone is offloaded to a local server or a remote cloud over the Internet. While the benefit of

faster processing can be generally expected, the impact on the energy efficiency of the smartphone seems less trivial because transferring the data needed for computation can also be very energy-consuming due to intensive communication.

A large body of work is focused on exploring energy-efficient computation offloading, and an overview can be found in [20]. Many projects [21], [22] assume a simple linear energy consumption model, based on which they explore the min-energy computation offloading scheme. Guo et al. also follow the linear energy consumption model and consider a more complex scenario where constraints on task dependencies and completion time deadlines are enforced [23]. Lyu provides a distributed integration architecture of the cloud, edge, and IoT devices, and propose a lightweight framework for energy-efficient selective computation offloading [24]. Offloading decision making for optimized energy efficiency has also been explored for specific applications such as real-time video analytics [25] and augmented reality [26].

In contrast, our proposal provides a general way to measure the energy consumption of low power devices and to quickly build a realistic energy consumption model using the measurement results. We are among the first to consider the energy consumption of the whole sensing-processing-actuating cycle of sensor-based applications instead of looking at only the processing part when making computation offloading decisions.

VII. Conclusion

We designed the Aves framework that allows developers to easily build context-aware applications in a heterogeneous environment using a unified programming API that follows a sensing-processing-actuating model. Aves reduces the application development complexity by allowing local and remote sensing, processing, and actuation on different categories of devices, such as the watch, the phone, and the cloudlet. The framework uses a decision-making engine based on an energy-prediction model that can automatically place the processing-task in an energy-efficient manner, given a scenario at hand. The energy-model is built based on a set of synthetic workloads that are measured offline once.

We validate the model and show that our linear regression based model is a good fit and is accurate for most scenarios. We observe that the battery percentage of a device has an impact on offloading decisions. Our framework significantly improves the application lifetime compared with different placement strategies. We show that the best choice estimated by the decision engine is the same as the one measured using a hardware power monitor for two real-world applications.

References

[1] A. Berger, A. Bischof, S. Totzauer, M. Storz, K. Lefeuvre, and A. Kurze, "Sensing home: Participatory exploration of smart sensors in the home," in *Social Internet of Things*. Springer, 2019, pp. 123–142.

[2] S. Wolfert, L. Ge, C. Verdouw, and M.-J. Bogaardt, "Big data in smart farming–a review," *Agricultural Systems*, vol. 153, pp. 69–80, 2017.

[3] Y. Sun, H. Song, A. J. Jara, and R. Bie, "Internet of things and big data analytics for smart and connected communities," *IEEE access*, vol. 4, pp. 766–773, 2016.

[4] K. Sakuma, A. Abrami, G. Blumrosen, S. Lukashov, R. Narayanan, J. W. Ligman, V. Caggiano, and S. J. Heisig, "Wearable nail deformation sensing for behavioral and biomechanical monitoring and human-computer interaction," *Scientific reports*, vol. 8, no. 1, p. 18031, 2018.

[5] F. Firouzi, A. M. Rahmani, K. Mankodiya, M. Badaroglu, G. V. Merrett, P. Wong, and B. Farahani, "Internet-of-things and big data for smarter healthcare: from device to architecture, applications and analytics," 2018.

[6] G. Lewis and P. Lago, "Architectural tactics for cyber-foraging: Results of a systematic literature review," *Journal of Systems and Software*, vol. 107, pp. 158–186, 2015.

[7] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham *et al.*, "Storm@ twitter," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014, pp. 147–156.

[8] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin *et al.*, "Apache spark: a unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[9] R. Kemp, "Programming frameworks for distributed smartphone computing," Ph.D. dissertation, Vrije Universiteit Amsterdam, 2014.

[10] N. Palmer, R. Kemp, T. Kielmann, and H. Bal, "Swan-song: A flexible context expression language for smartphones," in *Proceedings of the Third International Workshop on Sensing Applications on Mobile Phones*. ACM, 2012, p. 12.

[11] M. Solutions, "Monsoon power monitor," 2019. [Online]. Available: https://www.msoon.com/

[12] Q. Technologies, "Trepn power profiler," 2019. [Online]. Available: https://developer.qualcomm.com/software/trepn-power-profiler

[13] R. B. Das, N. V. Bozdog, M. X. Makkes, and H. Bal, "Kea: A computation offloading system for smartphone sensor data," in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2017, pp. 9–16.

[14] G. R. Ruiz and C. F. Bandera, "Validation of calibrated energy models: Common errors," *Energies*, vol. 10, no. 10, p. 1587, 2017.

[15] B. Ur, M. Pak Yong Ho, S. Brawner, J. Lee, S. Mennicken, N. Picard, D. Schulze, and M. L. Littman, "Trigger-action programming in the wild: An analysis of 200,000 ifttt recipes," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, 2016, pp. 3227–3231.

[16] Kitxoo, "Tasker for android," 2019. [Online]. Available: https://tasker.joaoapps.com/index.html

[17] T. A. S. Foundation, "Apache edgent," 2016. [Online]. Available: http://edgent.apache.org/

[18] H. Debnath, N. Gehani, X. Ding, R. Curtmola, and C. Borcea, "Sentio: Distributed sensor virtualization for mobile apps," in *2018 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, 2018, pp. 1–9.

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

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

[21] W. Zhang, Y. Wen, and D. O. Wu, "Energy-efficient scheduling policy for collaborative execution in mobile cloud computing," in *INFOCOM*, 2013, pp. 190–194.

[22] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, and Y. Zhang, "Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks," *IEEE Access*, vol. 4, pp. 5896–5907, 2016.

[23] S. Guo, B. Xiao, Y. Yang, and Y. Yang, "Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing," in *INFOCOM*, 2016, pp. 1–9.

[24] X. Lyu, H. Tian, L. Jiang, A. V. Vinel, S. Maharjan, S. Gjessing, and Y. Zhang, "Selective offloading in mobile edge computing for the green internet of things," *IEEE Network*, vol. 32, no. 1, pp. 54–60, 2018.

[25] L. Zhang, D. Fu, J. Liu, E. C. Ngai, and W. Zhu, "On energy-efficient offloading in mobile cloud for real-time video applications," *IEEE Trans. Circuits Syst. Video Techn.*, vol. 27, no. 1, pp. 170–181, 2017.

[26] K. Chen, T. Li, H. Kim, D. E. Culler, and R. H. Katz, "MARVEL: enabling mobile augmented reality with low energy and low latency," in *SenSys*, 2018, pp. 292–304.