Predictive Resource Management in Energy-constrained Embedded Systems

Simone Crippa*, Giuseppe Massari†, Federico Reghenzani†, Michele Zanella†, William Fornaciari†
*DEIB, Politecnico di Milano, Italy, - Email: simone3.crippa@mail.polimi.it
†DEIB, Politecnico di Milano, Italy, - Email: [name].[surname]@polimi.it

***Abstract*—The current trends in Internet of Things (IoT) lead to the deployment of low-power devices covering a wide range of application scenarios. These devices have the goal of executing simple tasks, automatically, usually with strict requirements in terms of space and cost. Typically, these devices have to rely on batteries or by harvesting energy devices (e.g., solar panels), in order to operate. On the other hand, IoT devices may be equipped with powerful multi-core CPUs to achieve performance goals, making the management of the energy budget a challenging task. This requires the development of an effective management system, that takes into account current and future energy budget availability, to dynamically bound the actual allocation of processing resources. Specifically, when exploiting solar panels for power supply, we can leverage on the weather forecast, to estimate the availability of energy in the near future. This paper introduces a predictive energy budget management system, targeting multi-core based embedded platforms. Thanks to both local and large-scale weather information, our solution aims at predicting the future incoming power and, accordingly, tuning the exploitable performance level to keep the system running under any environmental condition.**

***Index Terms*—IoT; energy-aware; embedded system; power management; machine learning; fault detection; solar energy; scheduling; multi-core**

I. Introduction

The use of Internet of Things (IoT) devices is increasing in many domains, including industry, automation, communication, medicine and transportation. Many of the applications in these domains require such devices to be small and constrained with respect to the energy and power usage. This in fact, represents the main performance scaling barrier of such systems. The challenge is indeed to maintain the ratio of power consumption over performance constant. However, many IoT devices run on batteries, making the energy problem also critical. An effective approach is to introduce *energy-awareness* in the system operation modes: the device manages the available energy budget by constraining its throughput or set of features, leveraging the underneath hardware hooks to control the power consumption over the time. This chase for energy efficiency is also caused by a stagnation in battery technology. Although both industrial and scientific communities are focused on improving the energy density of batteries, these increases are not sufficient to satisfy the constantly increasing demand of performance. The energy problem is even exacerbated when dealing with energy harvesting systems [18], i.e. systems relying on an electrical power source self-harvested from the environment, such as a wind turbine or a solar panel. The constraint on the energy budget availability, in these devices, is given by the limited battery capacity and the future quantity of available input power, which cannot be easily predicted. In fact, the external power sources are unreliable, since they are heavily dependent on weather conditions, yielding to high volatility in the exploitable energy budget. Energy harvesting systems are usually auto-sufficient, i.e. they are designed to guarantee the highest uptime possible with the least human intervention possible.

A. The Role of the Resource Manager

In the design of an embedded computing system, the exploitation of energy harvesting part introduces at least two main points to address:

1) How to define the size of the power supply components (capacity of the battery, size of the solar panel, etc…)?
2) How to choose the computing platform (e.g, microcontroller-based or System-on-chip) the system will be based on?

For the first point, we may need to take into account logistics aspects like the constraints related to the place for the physical installation of the system, the transportation, the maintenance and last but not least, the costs. The choice of the platform instead, is partially driven by the application requirements and partially constrained by the choices made for the first point. This means that we must manage the trade-off between computational capabilities and power consumption, given the aforementioned constraints.

Considering the very common approaches and given the system requirements and constraints, we typically have two design options: 1) to choose the power supply components, that would guarantee the system to remain up and running in most of the worst possible conditions; 2) to choose the computing platform, trying to minimize the power consumption requirements, such that also the size of the power supply components could be minimized.

Such static approaches are often very conservative, hence sub-optimal. We believe, in fact, that through a more dynamic energy-aware adaptive approach, we can minimize the size of the power supply components, while maximizing the capabilities of the computing platform. This would be made possible by a *resource manager*, in charge of varying the availability of resources (e.g, CPU cores), on the basis the current status

of the overall system (e.g., the energy-budget availability) [1], [2]. The resource manager, therefore, will be responsible of pursuing a two-fold objective: 1) to guarantee the running status of the system and 2) to control the level of performance that can be delivered by the computing platform. This means that, if we consider an embedded system supplied by a combination of batteries and energy-harvesting solutions, the resource manager must consider the energy budget available and the amount required to run the applications, to keep the system still running. As it should easy to guess, a similar approach requires periodical predictions of the energy-budget availability, such that the resource manager could make the best possible choices. The goal of this work is to investigate the possibilities offered by this approach.

B. A Motivational Use-case: A Transponder Tracking System

The transponder is an on-board device of an aircraft that periodically broadcasts the position and other flight information. The ADS-B system is widely used by almost all commercial airplanes for airborne collision avoidance or for tracking purposes. However, an ADS-B transponder is usually too expensive and too much power-hungry for gliders and ultralight aircraft. In this cases, a preferable choice is to use less common technologies enabling a simpler anti-collision system. The tracking of such an aircraft, from the ground, requires to build a dedicated network of receivers, because neither the government or the amateur network for ADS-B can be exploited. Flying clubs, for example, face the necessity to track their small aircraft, without having the financial capability of building a network of receivers. The solution we are currently studying in the area of Lombardy (Italy) is to place the receivers on the tops of few mountains to cover the largest possible area. The placement of such devices in these remote areas limits the possibility to use the electrical grid as a power source. Consequently, we decided to equip the devices with a battery and a solar panel to harvest energy. The energy source is then non-reliable for the reasons already explained. The resource management strategy presented in this paper has been created to guarantee the system uptime given the weather information.

C. Paper structure and contribution

In this work, we aim at presenting a predictive energy budget management system, targeting multi-core based embedded platforms. The system, governed by a resource manager, is able to: (a) predict the future incoming power based on a hybrid weather forecast mechanism, (b) control the performance capabilities of the computing platform, to satisfy the uptime requirements given the limited energy budget. The paper is organized as follows: in Section II, we present the available approaches related to the key points of our work, highlighting the difference and improvements. In Section III, we discuss the architecture of the systems we are targeting and our implementation. Then, in Section IV, we describe the models implemented for energy budget prediction, while in Section V, we discuss how our resource allocation policy exploits such predictions. Finally, in Section VI we show the experimental results obtained by deploying the system in a real environment, to conclude then in Section VII with the final remarks.

II. Related Works

This section presents some of the state-of-the-art approaches. First, we provide an overview of the works that already used weather forecasting to build energy budget prediction models. Then, the literature on energy-aware scheduling models and algorithms are presented. Finally, we discuss the advances explored by this paper.

A. Energy budget modeling

The approaches using the weather forecast to predict the amount of power harvested by a solar panel can be summarized in two classes. The first one considers the direct relationship between weather and output power. The second one, split it into two sub-problems: (1) to extract from the weather forecast the data that directly affect the power generations; (2) to use such data for carrying out the power predictions. We can also distinguish between Machine Learning (ML) and Time-Series (TS) based approaches.

In [9] the authors addressed the problem of predicting the charge level of the battery, according to the weather conditions, by using different ML models on historical data. In particular, they took into account some sun parameters and the battery technology characteristics. Thus, they approximated the charging curve and obtained the variation of charge, by measuring the battery voltage. To the contrary, in [15], authors adopted the problem-splitting approach by considering three sub-problems to: (1) model the solar panel efficiency; (2) infer the maximum output power of the panel; (3) penalize the maximum output of the solar panel using an external weather forecast provider. The final model carries out the power generated from the solar panel. With respect to [9], it does not consider the dependency on the battery, making the model independent from the battery parameters and technology. The same authors, in [16], used ML regression models, based on weather forecast data to predict the solar irradiance. The dataset is again based on historical data. However, the model presents several simplifications regarding the features considered, the output value and the learned function, with respect to the previous works. The model, in fact, does not consider the presence of an external battery or a solar panel. Differently from the previous ML-based works, authors in [4] described two TS-based prediction algorithms. They require data related to the energy flowing into the system to compute the solar panel output given the weather conditions. In particular, one of the algorithms improves the model accuracy, by taking into account the relationship between the solar irradiance and the weather conditions.

B. Energy-aware scheduling algorithms

As aforementioned, in our work the energy budget prediction obtained has to drive an energy-aware resource allocation or task scheduling policy. The preemptive scheduling

algorithm presented in [6] (called *Lazy Scheduling Algorithm*, LSA) is based on a static analysis of the tasks and the management of three system power configurations (called working modes). This means that we can apply the scheduler to real-time tasks for which arrival time, deadline and energy demand are known a priori. Moreover, in case of finite energy budget source, the algorithm needs to sample the input energy value for each execution. The algorithm, therefore, computes the schedule and set the working mode, taking into account the timing requirements and the energy availability. Authors of [14] improved this work, by introducing relevant changes to the energy management and reducing the computational complexity. Similarly, in [19] the authors presented an algorithm on the Early Deadline First (EDF) scheduler, which exploits Dynamic Voltage and Frequency (DVFS) to manage the performance-power trade-off. As it follows, the selected frequency affects the Worst Case Execution Time (WCET) estimation and the power consumption. Differently, in [13] authors proposed two solutions to schedule tasks assuming the presence of a rechargeable battery: a static solution using a worst-case scenario and a dynamic one, considering extra energy available when the worst-case does not occur. In [12], the authors dealt with worst-case energy consumption by directly estimating it from measurements. These probabilistic values are then used to schedule the task according to a mixed-criticality approach. Finally, in [8], a QoS-aware scheduler for harvesting systems uses both short- and long-term energy predictions in a two step algorithm. A first offline phase computes a minimized energy budget, based only on the long time predictions and the requested QoS level of the tasks. However, any error made by the prediction is not identified and mitigated, at run-time, since the budget value is fixed. The online scheduler takes the budget to pick the jobs to schedule or drop, minimizing QoS violations.

C. Discussion

Regarding the weather prediction model, our work follows the Sharma *et al.* approach [15] of splitting the problem into two steps, since we aim at treating separately the solar irradiance prediction and the effective estimation of the harvested power. For ML and TS methods instead, we cannot choose the most accurate method, but can make a comparison on input requirements and performance. In particular, the ML methods require an initial training phase and a data collection step, but a single sample of the current conditions is sufficient to compute a prediction. Conversely, TS models require constant logging as well as system energy measurements. Due to these considerations, we decided to choose a ML method. In this regard, we improved the [16] model with a new approach in the data generation phase, by exploiting local sensors to mitigate the location approximation error caused by weather providers. Furthermore, unlike [16], we maintain a general validity of the model removing hardware dependencies and avoiding over-simplifications. Regarding the scheduling and allocation policy instead, we consider an intermediate approach between [14] and [19], by exploiting DVFS and the heterogeneous architecture of the CPU of the computing system. Finally, in our experimental setup, similarly to [8], we tested the system in a real environment, considering the weather forecast as input for the energy budget computation and adapting the system performance limits, at run-time, accordingly.

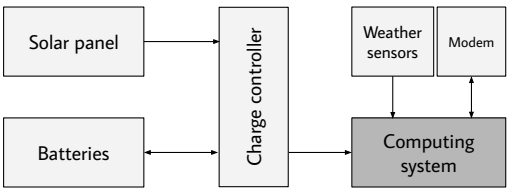


Fig. 1. Block diagram of the hardware system.

III. System Architecture

In this Section, we discuss the overall system architecture, by considering both hardware and software aspects. We briefly describe the implementation activities regarding the software part, while the details of the hardware system are left to the experimental section.

A. Hardware Architecture

In Figure 1, we sketched the typical structure of the embedded systems our work focuses on. The core is represented by a computing system, featuring a networking device (modem) for remote control and data transmission, other than a dedicated set of "weather sensors". The energy budget is defined by the power provided by a solar panel and a battery pack. A charge controller is then responsible of the process of battery recharging, other than regulating the power supply for the computing platform. As we said, in such a system, the energy budget is characterized by a certain degree of variability, given by the weather conditions, affecting the solar panel, and by the current level of charge of the batteries. On the system hand, the running tasks and applications may reasonably use a variable amount of resources (CPU, memory, etc...) over the time, which determines also the variability of the overall power consumption of the system.

Given the source of energy used to keep the system running, our idea is to exploit the weather forecast to estimate the available energy budget in the near future, and then dynamically bound the maximum amount of exploitable resources. This in order to balance between performance and power saving modes, to meet the application requirements on one side and preventing the system to run out of energy on the other one. More specifically, what we want to explore is the interplay of weather forecast information coming from a remote service on the Web and the data provided by weather sensors, locally connected to the system. This is motivated by the fact that the weather forecast services typically provide coarse-grained information, both from the spatial (city) and the temporal perspective (one prediction per hour), although obtained through accurate models and observations. The weather sensors instead, allow the system to 1) gather data with a much

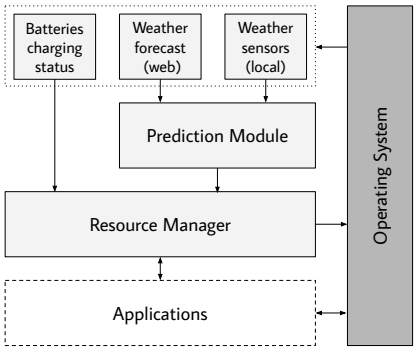


Fig. 2. Block diagram of the software stack of the system.

higher rate and 2) capture extremely local conditions, like, for example, transient clouds covering the sky and decreasing the solar irradiance for a few minutes. The objective is to demonstrate how, by exploiting these two sources, we can actually improve the accuracy of the energy budget estimation, to correctly drive the choices of the resource manager. More details about the hardware configuration of the system are provided in Section VI-A, where we describe the components of the actual setup built for the experiments.

B. Software Architecture

From the software perspective, the target system is characterized by a stack like the one shown in Figure 2. In the middle, we find the *Prediction Module* and the *Resource Manager*. The former takes as input the weather data, coming from the web-based forecast service and the local sensors, to predict the output power of the solar panel, as explained later in Section IV. The *Resource Manager* then, monitors the charging status of the batteries and take the predicted incoming power value to define the available energy budget. The "energy-aware" resource allocation policy, which is part of the manager itself, considers the energy budget value as input, along with the set of applications and, optionally, further system parameters. The policy enforces the assignment and sets the limits on the resource usage, by exploiting the interfaces provided by the underlying operating system. In this regard, we may have to choose between performing dynamic voltage-frequency scaling (DVFS) of the processing cores or setting an upper-bound on the number of exploitable cores.

C. Implementation

We implemented our experimental solution on a multi-core based embedded development board, for which we provided more details in Section VI-A. Most of the development effort has been focused on implementing the Prediction Module and extending the BarbequeRTRM run-time resource manager [3] for the following purposes:

1) performing the periodic monitoring of the status of the batteries (by accessing the Charge Controller interface);
2) gathering the output from the Prediction Module;
3) implementing the energy-aware resource allocation policy, described in Section V.

As shown in Figure 2, the Prediction Module retrieves the input data from a weather forecast service (on the web) and a set of local sensors. For the former, we chose *Weatherbit.io* [1]. The subscribed service allows us to receive an update per hour. The weather sensors instead are located on specific hardware modules, connected to the development board via I^2C bus and serial port and provide the following data:temperature [°C], humidity [%], visible light [lx] and pressure [hPa].

Finally, the resource allocation policy execution is triggered whenever an update on the energy budget value has been carried out.

IV. Energy-Budget Prediction Models

The energy-budget is determined by two main sources: the battery charge level and the power generated by the solar panel, in a given time frame. We assume that the charge controller provides us both values. However, the instantaneous values of power coming from the solar panel do not represent a very useful information, since the resource allocation policy needs to take decision considering a certain time horizon. Therefore, the first objective to achieve is to build a model which predicts the power generated by the solar panel from the weather conditions. Accordingly, assuming this forecast holding for a certain time frame, we can estimate an energy budget value for the resource allocation policy.

A solar panel generates power depending on the level of solar irradiance hitting its surface. Similarly to [15], we split the problem in two parts. First, we looked for a model to bind the weather data to the solar irradiance. Then, we built a second model to make explicit the relation between solar irradiance and the power output of the solar panel.

A. Solar Irradiance Model

While the weather forecast service directly provides instantaneous values of *solar irradiance* (SI), for the local sensors we need to infer it from the visible light value (L), by using the Formula from [11]:

$$SI\left[\frac{W}{m^2}\right] = 0.0079 \cdot L[lx] \tag{1}$$

This carries out an approximated yet acceptable value. The remaining sensors instead, *temperature* (T), *humidity* (h) and *pressure* (ρ), allows us to replace the usage of the *precipitation potential* as model feature, as proposed by Razi *et al.* in [10].

We are looking for a function $Y_1 = f_1(X_1)$, where Y_1 is the vector of the solar irradiance predictions, and X_1 is the vector of features, made as it follows:

$$X_1 = \{T, h, \rho, L\}$$

A worth remarking difference with respect to the model proposed by Sharma *et al.* [16] is that, in their case, the goal is to provide a long-sighted prediction of the solar irradiance value. Conversely, we aim at obtaining short-sighted predictions, that are used to correct the data provided by the

[1] https://www.weatherbit.io/

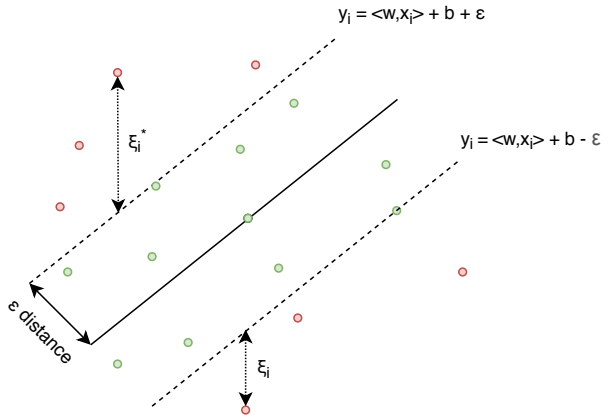


Fig. 3. Support Vector Regression optimization problem proposed by Kleynhans et Al. [7]

weather forecast service. Moreover, a further aspect to take into account for filling the dataset for the model construction, is that the weather sensors provide a new set of data samples every second. Such data are often characterized by the same values repeated over a long series of samples. This requires an aggregation step. Therefore, we increased the time-space between the samples in the dataset, by taking the mean value of the samples generated by the sensors every minute.

B. Power Generation Model

Once obtained the solar irradiance value, a further parameter affecting the solar panel performance is the relative position of the Sun. The most accurate value is given by the knowledge of the Sun incidence angle on the solar panel, computed from the panel inclination, the Sun zenith and the azimuth angles. An alternative solution (the one adopted), consists of including again the day of the year (D) and time of the day (t) in the features set, along with the solar irradiance (SI); while for the panel inclination we provided a constant value.

Therefore, in this case, given the vector Y_2 of generated power values, sampled from the panel, we are looking for a relationship such that $Y_2 = f_2(X_2)$, where X_2 is the set of features defined as it follows:

$$X_2 = \{SI, D, t\}$$

Given the small set of features and the low complexity of the problem, no feature selection was needed. This model is expected to implicitly capture two relationships, in order to predict the output generated power:

1) The solar panel characteristic, in particular its efficiency.
2) The relative position of the Sun with respect to day and time.

C. Support Vector Machines (SVM)

In both cases, we based the construction of the models on Support Vector Machines (SVM), which represent a good compromise between accuracy and computational requirements. We performed a preliminary experimental analysis on our hardware setup to evaluate the performance of such class of models. We found out that, for sizes of problem similar comparable to ours, we can obtain a prediction in tens of milliseconds with a memory occupancy of a few Megabytes.

More specifically, we used a Support Vector Regression model framework [17], based on a Radial Basis Function (RBF) kernel. Accordingly, the models parameters are obtained by solving the following optimization problem:

$$\min \ \tfrac{1}{2}||w||^2 + C \sum_{i=1}^{n} (\xi_i + {\xi_i}^*)$$
$$\text{subject to} \begin{cases} y_i - \langle w, x_i \rangle - b \leq \epsilon + \xi_i \\ \langle w, x_i \rangle + b - y_i \leq \epsilon + {\xi_i}^* \\ \xi_i, {\xi_i}^* \geq 0 \end{cases} \tag{2}$$

This problem, sketched in Figure 3, introduces the concept of *soft margin*. Looking at the figure, the n samples are represented by red or green circles. The red color indicates samples beyond the margin, while the green the ones within. Two lines, given by the functions y_i, define the margin. These functions are based on the dot product between the features weights (w) and the features values of the sample (x_i) plus b, the bias term. The model can avoid fitting a sample inside the margin, by paying a penalty, which is regulated through the C parameter: the larger the C the higher the penalization term related to the margin violation. For this reason, large values C can lead to overfitting. Conversely, a small value of C would lead to underfitting. Every violation of the margin corresponds to an ξ term related to the distance between the sample and the margin itself. The remaining parameters are:

- γ – which is related to the RBF based models only and represents the inverse of the standard deviation of the RBF kernel. γ defines an overall scale factor for the SVM notion of distance between two points. Therefore, it determines how far a support vector influences the decision boundary in its nearby neighbourhood.
- ϵ – which is the margin of error tolerated by the model.
- $\xi_i, {\xi_i}^*$ – the slack variables representing the distance of the sample beyond the margin ϵ, as shown in Figure 3.

These parameters, in particular C and γ, allow the model to be very flexible in the data fitting process. Finally, to verify the performance of the model we carried out a cross-validation phase, as explained in the next subsection.

D. Training and Validation

For the training and validation processes, we built the dataset by merging the weather data and the values of the power generated by the solar panel, read from the charge controller interface. The dataset has been populated with samples obtained by running the experimental system for several days, characterized by a certain variability of weather conditions. Then, we split it between training set and validation set. For both the models, we decided to use $\frac{2}{3}$ of the dataset for the training and the remaining $\frac{1}{3}$ for the testing. This is a reasonable compromise, given the size of our datasets, as the weather dataset included 75000 samples, while the

Model	% of SV	MSE	SCC
Solar irradiance	18.76	1651.32 $[W^2/m^4]$	0.968
Power generation	28.64	8.346 $[W^2]$	0.857

TABLE I
PERFORMANCE EVALUATION OF THE MODELS

power dataset was composed by 30000 samples. Moreover, we did not have to further split the set for cross-validation, since we exploited a K-Fold cross-validation approach, which includes a shuffling phase. To comply with the constraints given by the radial basis function kernel instead, we needed to scale the data, in order to shape all the features as Gaussian distributions with zero mean and unitary variance. Finally, we carried out the cross-validation phase, by using the *Grid Search* approach and providing the following vectors of values to the parameters:

- C values: [0.1, 1, 10, 100, 1000, 10000, 100000]
- γ values: [$\frac{1}{n*var(X)}$, $\frac{1}{n}$, 0.01, 0.1, 1, 10, 100]
- ϵ values: [0.001, 0.01, 0.1, 1, 10, 100]

where n is the size of X. This set of values has been experimentally chosen to maximize the exploration capabilities of the Grid Search. Table I reports the performance, in terms of *Mean-Squared Error* (MSE) and *Squared Correlation Coefficient* (SCC) and the percentage of support vectors (% of SV) obtained from the cross-validation and the training phase.

V. Energy-Aware Resource Allocation Policy

The resource allocation policy is based on the weather data coming from two sources: an online weather forecasting service (*remote*) and on the measurements provided by the sensors (*local*). The primary source used to predict the energy budget is the *remote* one, because its prediction is more reliable on the long-term. In fact, a weather service usually exploits large quantity of data, including satellite data, that consequently provides an overall picture of the atmospheric phenomena. To the contrary, the *local* data can be very useful to detect phenomena with respect to spatial and temporal locality, e.g. a small cloud over our system. The *local* data is then used to verify the prediction, based on *remote* data and, if there is a discrepancy, it is used for the next short time period to provide for the estimation error. The switching between the data sources is based on a threshold check. A coefficient Δ is computed as:

$$\Delta = \begin{cases} 1 & \text{if } |P_{\text{loc}} - P_{\text{rem}}|/P_{\text{loc}} < \chi \\ P^*_{\text{loc}}/P_{\text{rem}} & \text{else} \end{cases} \quad (3)$$

where P_{loc} is the power instantaneously read, P^*_{loc} is the power prediction based on local sensors, and P_{rem} is power prediction of the remote weather service. This coefficient Δ is then multiplied to the remote power prediction to adjust it. If $\Delta = 1$, then the remote prediction is considered valid and the local data ignored. The percentage threshold χ has been set according to empirical observations on the precision of remote weather service, as subsequently described in the experimental Section VI.

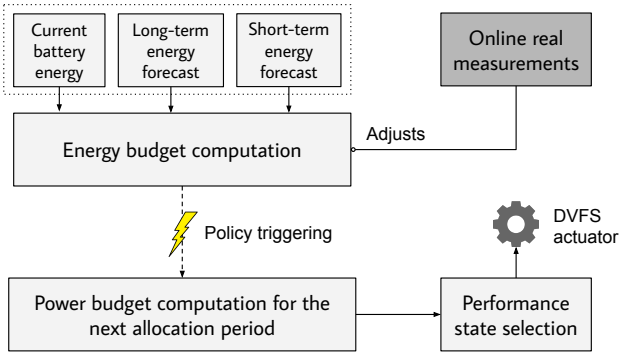


Fig. 4. Overall diagram of the resource management policy.

The resource management policy exploits such information to determine the DVFS performance state to use for the CPU cores. The Figure 4 summarizes the policy algorithm flow: in normal condition, the energy budget over the time period of 1 hour is predicted using the long-term information from the remote weather service and from the battery current state. Accordingly, the maximum power budget is computed over the 1 hour time interval and enforced via the DVFS mechanism (in our case the Linux *cpufreq* module). If the long-term prediction results unreliable ($\Delta < 1$), as explained in the previous paragraph, the short-term prediction is used and the power budget is computed over a time interval of 5 minutes, until the next weather information from the remote source has been made available.

The power budget is computed with the following expression: $P_{\text{budget}} = P_{\text{in}} - \varphi P_{\text{bat}}$, where P_{bat} is the power necessary to charge the battery - that depends on the current charge level - and $\varphi \in (0, 1]$ is the *reaction coefficient*. The value of φ tunes the policy behaviour: a more reactive approach by shifting the value towards 0, or a more conservative approach when shifting the value towards 1.

VI. Experimental Results

A. Experimental Setup

The deployment of our system in a real environment allowed us to learn about practical issues, test robustness and improve the reliability of the developed solutions, hence the design of the hardware setup was a critical step in our project. We started by focusing on the computing system, which was based on the ODROID-XU3 development board. This is a high-end embedded board equipped with a Samsung Exynos 5422 SoC, featuring a heterogeneous multi-core CPU, based on the big.LITTLE architecture (4x Cortex-A15 @ 2.0 GHz and 4x Cortex-A7 @ 1.4 GHz). The board includes also an INA231 on-board sensor which allowed us to monitor at run-time the power consumption of the main components of the SoC: the two CPU clusters (big and LITTLE) the RAM and the GPU. For the network connection, our goal was to emulate a real-world scenario. For this reason we plugged a Siretta ZEST-N-GPRS modem to fetch data from a weather forecast provider.

To add the ability of measuring environment parameters we added a set of sensors, including:

- A Silicon Labs Si1132, to measure UV Index, IR and visible light.
- A Bosch BMP180, to measure temperature and pressure.
- A Silicon Labs Si7020, to measure temperature, humidity and pressure.

To overcome the reliability issues given by extreme weather conditions, including high temperature and high relative humidity, we plugged the above sensors into a STM32 Nucleo-L412KB micro-controller based board. Finally, for the power supply part of the system: the core was represented by a solar panel, providing a peak power of 50 W, to charge a pack of two lead-acid Absorbent Glass Mat (AGM) batteries, with a capacity of 9 Ah each. To maximize the energy capacity of our system, the batteries were connected in parallel. The charging process of the battery pack was regulated through a charge controller (the Epsolar Tracer-A 10). This controller featured a RS485 interface, used to communicate with the computing system through the *MODBUS* protocol. This allowed us to monitor the battery pack state and the solar panel output power.

On the software hand, in order to exploit the multi-core architecture and emulate real-world workloads, we used the *Fluidanimate* parallel multi-threaded benchmark application from the PARSEC suite [5], to properly stress the processor.

Given this hardware and software setup, we defined two phases for this project, starting with a data collection phase required to build our prediction models and then a benchmark testing step, to evaluate the behaviour of our software solution.

B. Experimental Plan

Once the two aforementioned datasets has been built, as explained in IV-D, we conducted a series of experimental execution during the month of October 2019, when it has been possible to test the system under variable weather conditions. At this stage, we focused on two main goals:

1) making a comparison on the solar irradiance between the values provided by the weather services and data computed by local sensors;
2) observing the energy management capabilities and performance level delivery of the system.

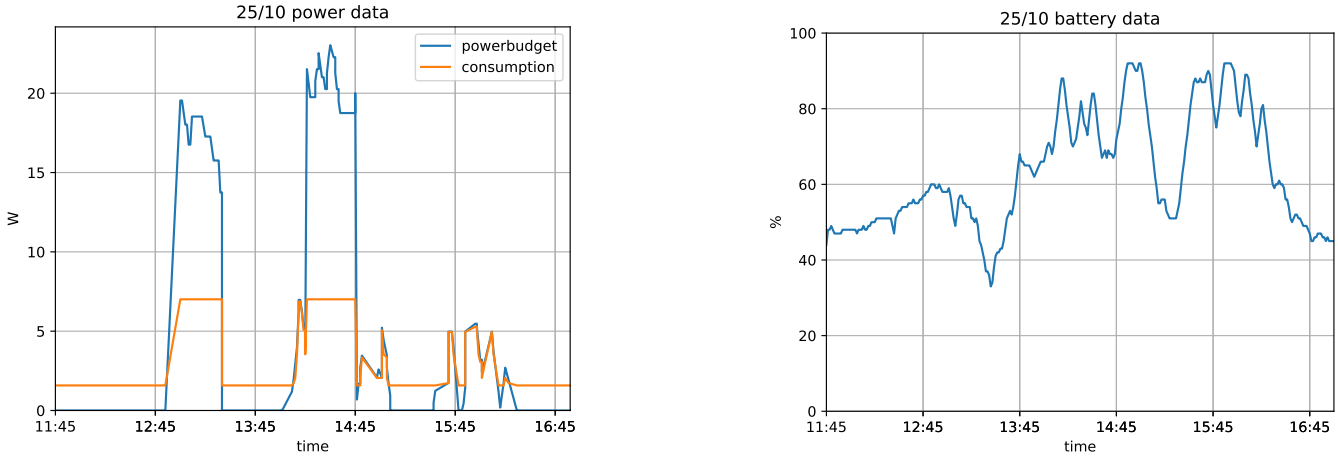
C. Irradiance Analysis

Since other solutions in the literature do not include local sensor to catch any anomaly in the weather prediction, we are interested in the number of prediction errors that we are able to detect, through the usage of the sensors. To this aim, we tested several violation thresholds in the range $\xi \in [0.25; 0.50; 0.75]$, which regulates how much the provided forecast of solar irradiance can deviate from the one measured through local sensors before considering it unreliable. The deviation probability is higher when lower values of threshold are used, with a direct impact on the number of energy budget updates.

In particular, we can define the *actual violations*, considering only the violations given by a misprediction of the weather service and not the ones given by the weather variability. Furthermore, we define the *false positives* as the percentage difference between all the violations and the actual ones. For this analysis, we used all data collected during August 2019 and October 2019. Data gathered by local sensors have been aggregated by computing the mean over a minute. The first goal is to explore which threshold value is able to detect the prediction errors of long-term forecast, taking into account variations of local weather due to temporary phenomena (e.g., clouds). In this regard, considering a $\xi = 0.5$ threshold, in August we reported a lower mean value (38.46%) of actual violations with respect to October (53.82%). Moreover, the lower percentage of false positives registered in October (5.08%), shows that in August solar irradiance prediction is more accurate. This leads us to consider the necessity of setting the threshold value on the basis of the season. The second goal is to determine the violation tolerance required to reduce the number of false positives. We observed that, during cloudy days, by increasing the tolerance level from $\xi = 0.25$ to $\xi = 0.50$, the false percentages of August (13.5%) and October (7.42%) are significantly reduced to 9.86% (from 13.5%) and 5.08% (from 7.42%) respectively. In these cases, an additional increase of threshold value leads to marginal improvements. In conclusion, we need to adapt the choice of the threshold not only considering the number of false positive, but also taking into account a priori information on the weather variability of the current season.

D. Policy Execution

In Figure 5a, we can observe the effects of the resource management action on the computing system. The blue line indicates the predictions of power budget, while the orange one is the actual consumption of the computing system, under the constraints set by the energy-aware resource allocation policy. For a matter of space, we report just one of the most interesting days, in which we experienced a certain variability of the weather conditions. We can see how, during the morning, the resource manager have aggressively forced the system to operate in a "power saving mode", minimizing the consumption in order to facilitate the recharging process of the batteries. This decision was also affected by the relatively low levels of solar irradiance. Lately, around 12:45 PM, the resource manager has relaxed the constraints (power budget above 15W), allowing the benchmark application to squeeze more performance from the platform (consumption of 6-7W). The system switched back to the power saving mode, after 1:15 PM, when the charge level dropped to 35%. Around this time, the weather conditions improved, allowing the batteries to recharge up to 90% and to the resource manager to allocate more processing power with respect to the previous hours. Again, at 2:45 PM the power saving mode was set to compensate the new drop of the battery charge level. This behaviour repeated itself until 3:15 PM. From that time point onward, the prediction module returned lower values of power budget, but thanks to more stable weather conditions, the system reported smoother battery charge variations until 4:30 PM. Finally, in



(a) Power budget management

(b) Battery-pack charge level

Fig. 5. October 25: Resource management effects, with the energy-aware resource allocation policy running on the experimental environment.

the afternoon, the resource manager forced again a power saving mode to retain the energy budget required to keep the system running during the night time and then, resume the workload execution the day after.

VII. Conclusions

We presented a novel approach to resource allocation for energy-constrained embedded computing systems, by exploiting weather forecast data from local sensors and from a remote forecast provider. The approach is based on the combination of the long-term prediction, provided by the remote weather forecast service, with the correction mechanism based on the local sensors readings. An energy-aware policy for a resource manager has been developed and tested to verify the ability of the system to survive even if the power source is not reliable. A set of experiments on a realistic setup demonstrated the adaptive behaviour of the system able to react to power source variability. Future works may extend our approach in many directions: for example, by improving the machine learning algorithms on the weather prediction, by searching novel approaches for weather sensor data fusion, or by developing and testing different resource allocation policies.

Acknowledgment

This research was partially supported by RECIPE H2020 EU project (grant no. 801137). We thank Mr. Denis Molinari for providing the experimental setup.

References

[1] Reliable power and time-constraints-aware predictive management of heterogeneous exascale systems. In *SAMOS*. ACM, 2018.

[2] The RECIPE Approach to Challenges in Deeply Heterogeneous High Performance Systems. *Microprocessors & Microsystems*, 2020.

[3] P. Bellasi, G. Massari, and W. Fornaciari. Effective Runtime Resource Management Using Linux Control Groups with the BarbequeRTRM Framework. *ACM Trans. Embed. Comput. Syst.*, 14(2):39:1–39:17, March 2015.

[4] C. Bergonzini, D. Brunelli, and L. Benini. Algorithms for harvested energy prediction in batteryless wireless sensor networks. pages 144–149, June 2009.

[5] Christian Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[6] L.Thiele C. Moser, D. Brunelli and L. Benini. Lazy Scheduling for Energy Harvesting Sensor Nodes. *IFIP International Federation for Information Processing*, January 2006.

[7] Tania Kleynhans, Matthew Montanaro, Aaron Gerace, and Christopher Kanan. Predicting top-of-atmosphere thermal radiance using merra-2 atmospheric data with deep learning. *Remote Sensing*, 9:1133, 11 2017.

[8] H. Kooti, N. Dang, D. Mishra, and E. Bozorgzadeh. Energy budget management for energy harvesting embedded systems. In *2012 IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 320–329, 2012.

[9] F. Kraemer, A. Ammar, D.and Bråten, N. Tamkittikhun, and D. Palma. Solar Energy Prediction for Constrained IoT Nodes Based on Public Weather Forecasts. 10 2017.

[10] Mohd Adib Mohammad Razi, Wardah Tahir, Noratina Alias, Lokman Hakim Ismail, and Junaidah Ariffin. Development of rainfall model using meteorological data for hydrological use. *International Journal of Integrated Engineering*, 5(1), Nov. 2013.

[11] A. Nouman, A. Chokhachian, D. Santucci, and T. Auer. Prototyping of Environmental Kit for Georeferenced Transient Outdoor Comfort Assessment. *ISPRS International Journal of Geo-Information*, 8:76, 02 2019.

[12] F. Reghenzani, G. Massari, and W. Fornaciari. A probabilistic approach to energy-constrained mixed-criticality systems. In *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6, 2019.

[13] Cosmin Rusu, Rami Melhem, and Daniel Mossé. Multi-version scheduling in rechargeable energy-aware real-time systems. *Journal of Embedded Computing*, 1(2):271–283, 2005.

[14] M. Severini, S. Squartini, and F. Piazza. Energy Aware Lazy Scheduling Algorithm for Energy-Harvesting Sensor Nodes. *Neural Computing and Applications*, 23, 12 2013.

[15] N. Sharma, J. Gummeson, D. Irwin, and P. Shenoy. Cloudy computing: Leveraging weather forecasts in energy harvesting sensor systems. pages 1–9, June 2010.

[16] N. Sharma, P. Sharma, D. Irwin, and P. Shenoy. Predicting solar generation from weather forecasts using machine learning. pages 528–533, Oct 2011.

[17] A. J. Smola and B. Schölkopf. A tutorial on Support Vector Regression. 2003.

[18] Sravanthi Chalasani and J. M. Conrad. A survey of energy harvesting sources for embedded systems. In *IEEE SoutheastCon 2008*, pages 442–447, 2008.

[19] Y. Xie, Zuodong W., and Shaojun W. An efficient algorithm for nonpreemptive periodic task scheduling under energy constraints. pages 128 – 131, 11 2005.