Energy-Resilient Real-Time Scheduling

Mahmoud Shirazi, Lothar Thiele, and Mehdi Kargahi, *Senior Member, IEEE*

Abstract—Embedded nodes in future cyber-physical systems are mostly self-powered, scavenging their required energy from the environment. The environmental sources of energy are usually variable, so that some prediction methods are employed to proactively adapt to the variable harvesting energy. However, prediction errors may surprise the system with some unpredicted changes, needing appropriate reactions. We consider an energy-harvesting real-time system with periodic tasks of multiple performance levels. An energy-resilient scheduler is proposed for the system to react to the unpredicted changes such that the system is survivable, recovers from such a change in a timely manner, and appropriately controls its performance degradation. After the recovery, however, the energy-resilient scheduler preserves the system survivability and maximizes its performance in a prediction time horizon, while it will be ready for another surprise. We provide some theoretical properties and a feasibility test which are used in the design of the energy-resilient scheduler. Our simulations show that the proposed resilient scheduler outperforms well-known performance maximization methods, effectively approximates the optimal solution, and reacts appropriately against surprises of high severity.

Index Terms—Resilience, real-time scheduling, energy harvesting, uncertainty, autonomous recovery

1 Introduction

Emerging concepts like ambient computing, more than ever emphasize key aspects of cyber-physical systems (CPSs) like operating in uncertain environments and the need to perpetual, autonomous, and unattended operation. The latter characteristics need survivability, namely uninterrupted operation of the system, although with the most limited (yet permitted) operating level. Therefore, such systems are to be energy-neutral through receiving their required energy from the environment. A conflict then appears between the critical situations the system may face with due to the uncertainty of available energy because of the environmental unforeseen changes and the demand for uninterrupted functionality. In fact, such systems need fast and successful recovery from such a critical situation with some controlled and minimal performance penalty. These properties can be summarized into a criterion called resilience, as defined below.

Definition 1. (Resilience [1], [2], [3]): *Resilience is fundamentally defined as "resuming the original shape or position after being bent, compressed, or stretched". More formally, resilience is the persistence of performability when facing changes; a resilient system must survive at some capacity, so that it will be able to recover autonomously.*

In this study, we discuss energy-resilience, as described in the following. Suppose a system of L performance levels, with an energy harvester and an energy storage unit, equipped with an energy predictor. An energy-resilient scheduler (we may call it a resilient scheduler throughout the paper) uses the predictions to adapt the system performance to the incoming energy from the environment, while trying to maximize the performance. However, due to the environmental uncertainties, the predictor may make some error once in a while, which comes as a surprise to the system. By surprise, we mean that the system estimation of the energy contents of the energy storage unit gets wrongly optimistic. A must towards energy-resilience is preserving the system survivability; further, the situation should be controlled by trading between its performance (functionality) level degradation for energy saving and the speed of recovery to the normal behavior; namely, a state at which the energy-resilient scheduler tries its best for performance level maximization, while being sufficiently conservative for another possible surprise. We say that the recovery is done when the energy storage returns to the right track, namely it appears that the system had never faced an energy shortage from the energy storage perspective; more precisely, as the time advances, in some instant on the timeline, the system finds itself according to the proper energy storage estimation at proper time.

Many energy-neutral embedded systems are real-time as well; therefore, even if the system faces an energy-critical situation, the timely responses of the system tasks should not be threatened. Therefore, some sort of conservative behavior with respect to spending the energy storage contents is needed, which in turn affects performance maximization during time intervals where the system is at the risk of facing a surprise. The time intervals that there is such a risk that the resilient scheduler should be more conservative depends on factors like how often the energy predictor makes an error and to what extent.

- *Mahmoud Shirazi is with the Department of Computer Science and Information Technology, Institute for Advanced Studies in Basic Sciences (IASBS), Zanjan 45137-66731, Iran. E-mail: m.shirazi@iasbs.ac.ir.*
- *Lothar Thiele is with the Computer Engineering and Networks Laboratory, ETH Zurich, 8092 Zürich, Switzerland. E-mail: thiele@tik.ee.ethz.ch.*
- *Mehdi Kargahi is with the School of Electrical and Computer Engineering, College of Engineering, University of Tehran, Tehran 31587-11167, Iran, and also with the School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran. E-mail: kargahi@ut.ac.ir.*

Manuscript received 31 January 2022; revised 1 August 2022; accepted 20 August 2022. Date of publication 29 August 2022; date of current version 13 December 2022.
(Corresponding author: Mahmoud Shirazi.)
Recommended for acceptance by D. Mosse, T. Chantem, and E. Bini.
This article has supplementary downloadable material available at https://doi.org/10.1109/TC.2022.3202754, provided by the authors.
Digital Object Identifier no. 10.1109/TC.2022.3202754

As an example, suppose an edge-computing application in a surveillance scenario, including an energy-autonomous mission-critical node with three performance levels. It harvests its required energy from a renewable energy source like light, temperature difference, vibration, or radio-waves. Suppose the system contains three different real-time tasks: Task 1 has to regularly read an image sensor of high-quality, low-quality, or still-image, corresponding to the system functionality levels. Task 2 performs video/image processing, which depending on the system functionality level takes high, medium, or low execution times. Task 3 transfers the results wirelessly to a central host. Certainly, every task of the system consumes more energy at a higher system functionality level.

In such an energy-harvesting scenario, the system normally tries to operate at a higher performance level. In case of a surprise in the amount of stored energy, the system may need to temporarily degrade its performance level to consume less computing power and energy, and recover from the prediction error. As a reasonable expectation, we tend to avoid the lowest level as much as possible; at the same time, we are to perform the recovery within a limited time, because another surprise is possible at some later time in a sporadic manner and the system should have sufficient stored energy to be able to perform resiliently against the next surprise. After the recovery, the system will be able to continue with the video surveillance and the performance maximization. However, according to the energy predictor which determines when the predictions are at the risk of another surprise, the resilient scheduler must decide when to perform conservatively.

The resilient scheduler is supposed to have two states: *Normal state* and *surprise state*. While there is no *surprise* in the amount of harvested energy, i.e., the energy received from the renewable energy source is as predicted, the resilient scheduler is in the *normal state*, where it tries to maximize the system performance with regard to the energy predictor error rate and the data over a given time horizon, and the initial amounts of energy in the storage unit. The *surprise*, however, may enforce the system to get away from the determined performances and the predicted amounts of stored energy. In such a scenario, our resilient scheduler goes into the *surprise state*, and tries to do recovery within some limited time.

Resilient scheduling in the context of real-time systems has not been investigated frequently yet. In fact, except a few studies [4], most researchers only take a few aspects of resilience into account, whereas in the current study, we simulateously consider survivability, performance, and time to recovery. Shirazi et al. [5] consider the performance maximization assuming that the prediction method is accurate. Stricker et al. [6] define robustness where the energy prediction method is inaccurate, and provide some trade-off between robustness and efficient use of resources in a real-time system. They define the degree of robustness as a comparison of the prediction error and the system performance without considering recovery against the prediction error. Anand et al. [7] consider the recovery time of real-time systems having redundant task options. They do not consider performance, nor do they consider system survivability. Regarding fault-tolerant real-time systems, some researches target the reliability and energy constraints of the systems by primary/backup [8], replication and N-modular redundancy [9], [10], checkpointing [11], and re-execution [12] solutions. However, such solutions are based on extra resource usage, including time, energy, or hardware. In the context of mixed-criticality systems, Sobhani et al. [13] simultaneously consider the time, energy, and QoS constraints into account. In [14], a method based on the dynamic voltage and frequency scaling (DVFS) is proposed to reduce the energy consumption of the mixed-criticality tasks. Cao et al. [15] consider the life-time of fault-tolerant mixed-criticality systems using DVFS. None of these works study simultaneous consideration of survivability, performance maximization, and time to recovery.

In this paper, we concentrate on a holistic view to an energy-resilient real-time system, via defining measures, giving some theoretical fundamental properties and conditions, proposing an energy-resilient scheduler, and evaluating its efficacy using extensive simulations to show that it effectively approximates the optimal solution and it can appropriately react against surprises of high severity. Overall, the main contributions of this paper can be summarized as:

- Formalizing the concept of energy-resilient real-time scheduling in the presence of energy prediction inaccuracy, with simultaneous consideration of system survivability, performance, and limited time to recovery;
- Defining a measure of resilience composed of several subsidiary measures;
- Providing theoretical results to determine the conditions for having an energy-resilient scheduler;
- Proposing a dual-state resilient scheduler for the system model considered in this paper; and
- Providing various simulation studies to show the efficacy of the proposed resilient scheduler.

The rest of this paper is organized as follows. Section 2 gives the model of system components, including energy supplier, energy consumer, and the resilient scheduler, and gives an intuitive problem definition. In Section 3, we define our terminologies as well as subsidiary and compositional measures and objectives. Section 4 provides some theoretical properties and conditions as prerequisites for proposing the resilient scheduler of Section 5. Section 6 shows the simulation results. Finally, Section 7 concludes the paper.

2 System Model

We consider a single processor real-time system, consisting of an energy harvester (of a variable energy source), an energy storage unit (e.g., a super-capacitor or battery), an energy consumer (the processor which runs the real-time tasks), and a resilient scheduler (RS). We refer to the combination of energy harvester and energy storage unit as energy supplier. Below, we describe the system components and then give an intuitive problem definition.

2.1 Energy Consumer

The energy consumer consists of a set of n real-time independent periodic tasks $\Gamma = \{\tau_1, \ldots, \tau_n\}$, where τ_i is shown as $(\langle e_{iL}, \ldots, e_{i1}\rangle, \pi_i, \langle pow_{iL}, \ldots, pow_{i1}\rangle)$, $1 \leq i \leq n$, in which π_i

denotes the task period with an implicit relative deadline, and $\Pi = LCM(\pi_i)$, $1 \leq i \leq n$, is the length of the task set hyperperiod. Each task τ_i has L operating levels; the vector $\langle e_{iL}, \ldots, e_{i1} \rangle$ shows the execution times corresponding to the operating levels, where e_{il} shows the task execution time at operating level l. Also, the element pow_{il} of the vector $\langle pow_{iL}, \ldots, pow_{i1} \rangle$ shows the worst-case dynamic power consumption of the system when it executes task τ_i at operating level l. Further, we denote the static power consumption of the system as pow_s. More precisely, when a task τ_i is running at performance level l, the power consumption of the system is $pow_{il} + pow_s$; otherwise, it is equal to pow_s. Higher performance levels provide either more functionality or more quality, however, at the expense of more energy consumption and more execution time, namely $0 < e_{il}pow_{il} \leq e_{il'}pow_{il'}, \forall 1 \leq l < l' \leq L$. pow_{il} includes computation/communication power consumption of task τ_i at performance level l. We say that the system is *survivable* if all jobs of each task τ_i, $1 \leq i \leq n$, are executed for their minimum execution time of e_{i1} with no energy shortage, and completed before their deadline.

Within each hyperperiod $h \geq 0$, every task τ_i, $1 \leq i \leq n$, is run in performance level l (determined by the RS), i.e., every job of the task takes e_{il} to execute, and the system performance level is as $Perf(t) = l$ for $h\Pi \leq t < (h+1)\Pi$. We note that, by conservative reservation of some amounts of energy in the storage unit, we can guarantee the planned performance level for the hyperperiod (as will be described in Section 4). In case that even $l = 1$ cannot be planned and guaranteed for the hyperperiod (namely, at least one job of a task τ_i cannot successfully be executed for e_{i1} and completed before its deadline), then $Perf(t) = -\infty$ for $h\Pi \leq t < (h+1)\Pi$. More formally, we define the system performance at time t as follows:

$$Perf(t) = \begin{cases} l, & \text{if performance } l \in [1, L] \text{ is planned} \\ & \text{and guaranteed for hyperperiod } \lfloor \frac{t}{\Pi} \rfloor . \\ -\infty, & \text{otherwise} \end{cases} \quad (1)$$

2.2 Energy Supplier

The energy supplier consists of an energy harvester and an energy storage unit. The energy storage unit is supposed ideal, with the nominal capacity of E_{max}. $E(t)$ shows the amount of stored energy in the unit at time t. The amount of energy in the storage unit at the start of hyperperiod h is thus $E(h\Pi)$. The energy harvester charges the energy storage unit with some harvesting rate (power) in the range of $[HP_{min}, HP_{max}]$ from a renewable energy source.

During a hyperperiod h, the harvesting rate jitters in a narrow range of $[hp_{min}, hp_{max}]$, where $HP_{min} \leq hp_{min} \leq hp_{max} \leq HP_{max}$. However, the actual harvesting rate of that hyperperiod, denoted by R_h, is represented as a constant value equal to the average harvesting rate of that hyperperiod, calculated through dividing the harvested energy of that hyperperiod by Π. We also suppose an energy predictor that gives a pessimistic forecast $\hat{R}_h = hp_{min}$ of the harvesting rate as a safe prediction. If $R_h \geq \hat{R}_h$, we say that the prediction is correct; however, the predictor, according to its specifications, may make an error with a maximum of $\epsilon_{max} \in (0, 1]$ once in a while, where $(1 - \epsilon_{max}) < \frac{R_h}{\hat{R}_h} < 1$. How much narrow $[hp_{min}, hp_{max}]$ is, how large ϵ_{max} is, and how often such a prediction error may happen depend on the source of energy and the energy predictor specifications. The relatively short length of hyperperiods in many applications of embedded systems (e.g., [16]) helps in having more deterministic predictors from these aspects.

If the hyperperiod length is relatively short, and the energy source (like solar) has no significant changes in the short time intervals, we can calculate some conservative prediction of the harvesting rates at the hyperperiod boundaries by using the minimum of the predictions overlapping the hyperperiod. We do so for a time horizon of H hyperperiods. By analyzing the energy predictor behavior, we can find the minimum distance between consecutive prediction errors; by converting the distance to the granularity of hyperperiod length, we obtain Δ_s, representing that two consecutive errors of the energy predictor may happen at least Δ_s hyperperiods apart, where we suppose $\Delta_s < H$. Thus, the values of Δ_s and H are given depending on the nature of energy source and the specifications of energy predictor, with regard to the system hyperperiod. In Section 3, we discuss Δ_s in more detail.

Suppose a prediction time horizon of H hyperperods. At the start of hyperperiod $h \geq 0$, the predictor gives $\hat{R}_h, \ldots, \hat{R}_{h+H-1}$, each corresponding to the prediction of one of the H further hyperperiods. Similarly, at the start of hyperperiod $h+1$, the next predictions will be $\hat{R}_{h+1}$, $\ldots, \hat{R}_{h+H}$. Namely, at the start of hyperperiod $h+1$, the system has the following new information: The prediction of hyperperiod $h+H$, and the actual energy in the storage unit at the end of hyperperiod h, namely $E((h+1)\Pi)$. To take care of the static power consumption of the system, we should subtract pow_s from the predicted and actual harvesting rates. However, for the sake of better readability of the formulations, we assume $pow_s = 0$ in the rest of this paper, except in Section 6.3 where non-zero pow_s is considered for the provided case-studies.

2.3 RS and Intuitive Problem Definition

In this paper, two schedulers are employed: A *resilient scheduler (RS)* and an energy work-conserving *task scheduler*.[1] The former determines the performance level of the system at the start of each hyperperiod; then, the latter runs the tasks according to some priority order within the hyperperiod.

Suppose the system is at the start of hyperperiod h. Given the predicted harvesting rates, we define $\hat{\zeta}(h) = \langle \hat{E}((h+1)\Pi), \ldots, \hat{E}((h+H)\Pi) \rangle$, where $\hat{E}(t)$ is the amount of energy predicted to be in the storage unit at time t. Therefore, $\hat{\zeta}(h)$ is a vector including the amounts of predicted energy in the storage unit at the hyperperiod boundaries for a time horizon of H hyperperiods. While there is no prediction error, this vector is updated at the start of every further hyperperiod with the prediction of a new hyperperiod in the prediction time horizon; however, if the RS encountered a prediction error with regard to hyperperiod h, it makes a

1. To take the time and energy overheads of the resilient scheduler into account, one may consider the scheduler as an additional task of the highest priority in the system model. The WCET and power consumption of this task is considered the same for all the system performance levels.

copy of $\hat{\zeta}(h)$ for some further usage, and then performs the above-mentioned update.

We define another vector $\zeta(h)$ of length H, whose values are set to zero initially. While there is no prediction error, the RS does not use this vector. However, if there is a prediction error at hyperperiod h, under specific circumstances which will be discussed later, the RS will update the first element of $\zeta(h)$ by the actual energy of the storage unit; namely, at the end of hyperperiod h, i.e., at time $(h+1)\Pi$, where the prediction error is detected, the RS only has the first element, so it updates the vector as $\langle E((h+1)\Pi),\, 0, \ldots, 0)\rangle$. Note that we have $E((h+1)\Pi) < \hat{E}((h+1)\Pi)$ due to the prediction error. Then, at the end of hyperperiod $h+1$, the RS updates $\zeta(h)$ as $\langle E((h+1)\Pi), E((h+2)\Pi),\, 0, \ldots, 0)\rangle$. Accordingly, we give an ordering between $\zeta(h)$ and the aforementioned stored copy of $\hat{\zeta}(h)$ by the following definition.

Definition 2. *Suppose vectors $\zeta(h)$ and $\hat{\zeta}(h)$. We denote $\zeta(h) \geq \hat{\zeta}(h)$, if $\exists k \in [1, H]$ such that $E((h+k)\Pi) \geq \hat{E}((h+k)\Pi)$.*

This definition is used in Section 3 for defining the time to recovery from a prediction error of hyperperiod h. In fact, the RS continues to update $\zeta(h)$ until it finds $E((h+k)\Pi) \geq \hat{E}((h+k)\Pi)$ for some $k \in [2, H]$, and then stops using $\zeta(h)$.

The RS is either in *normal state* or in *surprise state*. In both states, at the start of each hyperperiod, say h, it determines a feasible performance level, and starts to perform accordingly. The default state of the RS is *normal state*, where according to the energy predictions and the amount of energy in the storage unit, it plans for the performance levels of the system in the time horizon of H hyperperiods, with the goal of maximizing the system performance; according to the plan, it updates $\hat{\zeta}(h)$ and takes care of the system survivability as well, to guarantee that any task (and thus, the system) will not face energy shortage to operate in its minimum performance level $l = 1$ in the prediction time horizon, so that it can preserve $Perf(t) \geq 1, \forall t \geq 0$.

If the amount of harvested energy is according to the predictions, the RS performs as mentioned above. However, if it is less than what is predicted, the system may deviate from its usual plan, namely some performance levels may not be achievable or the system survivability may be threatened. In such a case, the RS goes to *surprise state*; where, it tries to return to the conditions that the actual amount of energy storage contents be equal to or greater than the predicted amount before the surprise, i.e., to be in accordance to Definition 2. This should be guaranteed to be done within some specified time frame. Then, the RS can again behave normally.

As a simple example, Fig. 1a shows the situation that the RS is in *normal state* at the start of hyperperiod h, and it has some predictions about the amounts of energy in the storage unit. In such a case, the predicted amounts of energy in the storage unit at the hyperperiod boundaries are $\langle 5, 6, 3, 1, 8\rangle$. Fig. 1b, however, shows the case where there is a surprise in hyperperiod h, so that the harvested energy is less than the prediction at the end of the hyperperiod (it is $1 < 5$). In such a case, disregarding how the RS has determined the performance levels, according to the actual amounts of energy in the storage unit at the hyperperiod boundaries, i.e., $\langle 1, 3, 4, 0, 0\rangle$, the RS has recovered from the surprise at the end of hyperperiod $h+2$, where $4 > 3$. Note that, Fig. 1b illustrates the storage unit state at time $t = (h+3)\Pi$, i.e., at the end of hyperperiod $h+2$; then, only the actual amount of energy in the storage up to that time is known; afterwards, the RS can forget that a surprise has happened, because it had supposed before the surprise that at the end of hyperperiod $h+2$ it has 3 or more units of energy, which it has been realized when it reaches there. Therefore, it can again behave normally.

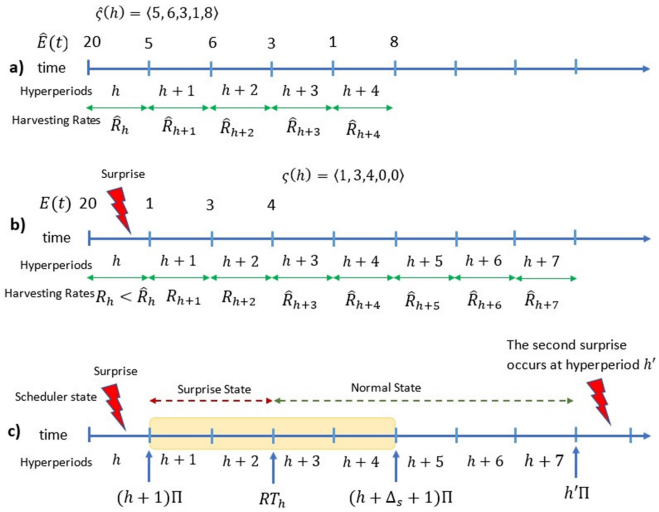

Fig. 1. The RS states and storage unit levels ($H = 5$, $\Delta_s = 4$): a) In *normal state*, b) In *surprise state*, c) Between two consecutive surprises.

To be more precise with respect to the RS and the problem considered in this study, more details of *normal state*, *surprise state*, their performances, survivability, and recovery, along with the related discussions and definitions, are needed, which are provided in the next section. Afterwards, the problem definition is also augmented by some additional formalism.

3 Definitions and Measures

Regarding the system model presented in the previous section, we face an *unpredicted change* when the actual amount of harvested energy is different from the prediction. Although the unpredicted change can result in either more or less harvested energy than the prediction, this paper focuses on the latter, which may surprise the system by putting it into some bad situations. We study the system resilience in such a situation, and propose a RS to deal with the *unpredicted changes* with attention to three concepts: survivability, time to recovery, and system performance. Below, we detail these concepts through some definitions.

Definition 3. (Survivability)*: The system is survivable at time T if for all $t \in [0, T]$, we have $Perf(t) \geq 1$.*

Definition 4. (Surprise)*: A surprise of $\epsilon \leq \epsilon_{max}$ in hyperperiod h is an error in $\hat{R}_h$ due to a harvesting rate R_h less than prediction such that $R_h = (1-\epsilon)\hat{R}_h$, which leads to have less than the predicted amounts of energy in the storage unit, namely $E((h+1)\Pi) < \hat{E}((h+1)\Pi)$. Such a surprise is detected at the end of hyperperiod h.*

In fact, a surprise is recognized by finding an energy shortage in the storage unit with respect to the prediction (Note that a surprise may happen at anytime within a

hyperperiod, however, since the proposed resilient scheduler conserves sufficient energy in the storage unit at the start of each hyperperiod to successfully run the system at the designated performance level, the surprise has no negative impact on the system behavior during the hyperperiod.). More precisely, as the RS plans for hyperperiods $h+1$ to $h+H$ according to the actual amounts of energy in the storage unit at the start of hyperperiod $h+1$, having $E((h+1)\Pi) \geq \hat{E}((h+1)\Pi)$ despite $R_h = (1-\epsilon)\hat{R}_h$ is interpreted as no surprise within hyperperiod h.

Since the RS decision on the performance levels is only made at the hyperperiod boundaries , if there is a surprise within hyperperiod h, the RS gets aware of that at time $(h+1)\Pi$, i.e., at the end of the hyperperiod, when the value of $E((h+1)\Pi)$ gets known. Then, the RS reaction depends on the relative amount of shortage, namely it depends on the *severity* of the surprise, defined as

$$S_h = \frac{\hat{E}((h+1)\Pi) - E((h+1)\Pi)}{\hat{E}((h+1)\Pi)}, \tag{2}$$

where $\hat{E}((h+1)\Pi)$ is the amount of energy predicted to be in the storage unit and $E((h+1)\Pi)$ is the actual amount of energy in the storage unit at the end of hyperperiod h. We only define the severity of a surprise upon detecting the surprise; thus, according to (2), $0 < S_h \leq 1$, where a larger value of S_h indicates a higher severity. We can also write

$$S_h \leq \frac{(\hat{R}_h - R_h)\Pi}{\hat{E}((h+1)\Pi)} = \frac{\epsilon \hat{R}_h \Pi}{\hat{E}((h+1)\Pi)}. \tag{3}$$

Since the root cause of the surprise is the error in the harvesting rate prediction, the right hand side of Inequality (3) is equal to the left hand side only when the storage unit capacity is unlimited; otherwise, namely if there is a maximum capacity of E_{max}, some harvesting energy may be wasted due to energy overflow if the harvesting rate is according to the prediction, so that $\epsilon \hat{R}_h \Pi$ is an upper bound on the storage unit energy shortage. We suppose that the RS takes care of the maximum possible error ϵ_{max} of the harvesting rate prediction method in a manner that $\epsilon_{max}\hat{R}_h\Pi \leq \hat{E}((h+1)\Pi)$ is always valid; therefore, even in the presence of the maximum surprise, the schedule of hyperperiod h at the pre-planned performance level of l can successfully be completed with support of the energy in the storage unit. This is considered by the RS through appropriate performance level selection for H hyperperiods.

Upon a surprise in hyperperiod h, a recovery to some target level of energy storage unit is expected. The target is determined based on the predicted storage unit state vector when there is no surprise; more precisely, we use $\hat{\zeta}(h)$ as our target for the recovery, as described in the following. The RS constructs the storage unit state vector of $\zeta(h) = \langle E((h+1)\Pi), \ldots, E((h+H)\Pi)\rangle$ element-by-element (which at the first step we only have the first element, i.e., $E((h+1)\Pi)$), and plans in a manner to reach $\zeta(h) \geq \hat{\zeta}(h)$ (see Definition 2), where we say that the recovery has been done. The time of this recovery certainly depends on the way that the RS makes its decisions, which in turn may depend on S_h. Formally, the recovery time RT_h against the surprise of hyperperiod h is calculated as

$$RT_h = \min_{2 \leq k \leq H} \{(h+k)\Pi | E((h+k)\Pi) \geq \hat{E}((h+k)\Pi)\}. \tag{4}$$

As we mentioned earlier in Section 2, $\Delta_s\Pi$ is the shortest interval between any two consecutive surprises. To avoid suffering from accumulated surprises, the recovery should be done before or at $(h+\Delta_s+1)\Pi$, namely with a time to recovery of at most $\Delta_s\Pi$. Section 4 explains how our method does such a guarantee, i.e., it preserves $RT_h \leq (h+\Delta_s+1)\Pi$ for a surprise in hyperperiod h. The *normalized time to recovery*, denoted by $NTTR_h$, is calculated as

$$NTTR_h = \frac{RT_h - (h+1)\Pi}{\Delta_s \Pi}. \tag{5}$$

Since the recovery is done at the hyperperiod boundaries, and according to (4) and the maximum permitted recovery time mentioned above, we have $\frac{1}{\Delta_s} \leq NTTR_h \leq 1$. When the surprise is of high severity, having a large $NTTR_h$ is a reasonable expectation from the RS. Otherwise, it is expected that the RS be able to reduce $NTTR_h$.

Before providing our target measure of resilience, the following definitions are needed:

Definition 5. (Surprise State)*: The* surprise state *is a RS state that the recovery has not been done. Given that a surprise has been happened in hyperperiod h, in* surprise state*, the RS tries to minimize $NTTR_h$ and maximize the performance within the time horizon of $[(h+1)\Pi, RT_h]$; which in turn preserves the system survivability.*

Definition 6. (Normal State)*: The* normal state *is a RS state that the recovery has been done. In* normal state*, the RS tries to maximize the system performance (and hence, to preserve the system survivability) within a time horizon of H hyperperiods.*

According to the RS states, the performance of the system is provided with a *state-based* definition, via separating two measures of *surprise performance* and *normal performance*.

Definition 7. (Surprise Performance)*: Suppose a surprise has happened in hyperperiod h, and RT_h is when the recovery is done and the system goes to* normal state*. The normalized performance during the* surprise state*, i.e., in the time interval of $[(h+1)\Pi, RT_h]$, is calculated as follows (see Fig. 1c):*

$$SurprisePerformance_h = \frac{\int_{(h+1)\Pi}^{RT_h} Perf(t)dt}{L(RT_h - (h+1)\Pi)}, \tag{6}$$

where L is the maximum performance level of the system. Therefore, we have either $\frac{1}{L} \leq SurprisePerformance_h \leq 1$ or $SurprisePerformance_h = -\infty$.

Definition 8. (Normal Performance)*: Suppose RT_h is the recovery time from the surprise of hyperperiod h. Afterwards, either we have another surprise or we have no further surprise. In the former case, $(h'+1)\Pi$ is the time of recognizing the surprise, where $h' - h > \Delta_s$. In the latter case, $(h'+1)\Pi$ is the end of the last hyperperiod of the system. Thus, the RS is in the* normal state *in the interval of $[RT_h, (h'+1)\Pi]$. The normalized performance during the* normal state *is then calculated as (see Fig. 1c)*

$$NormalPerformance_h = \frac{\int_{RT_h}^{(h'+1)\Pi} Perf(t)dt}{L((h'+1)\Pi - RT_h)}. \tag{7}$$

Therefore, we have either $\frac{1}{L} \leq NormalPerformance_h \leq 1$ *or* $NormalPerformance_h = -\infty$.

The last measure that we define is *Resilience*, which is defined as a composition of *surprise performance*, *survivability*, and *normalized time to recovery*, with regard to Definition 1

$$Resilience_h = \frac{SurprisePerformance_h}{NTTR_h \times \Delta_s}. \quad (8)$$

Therefore, we have either $\frac{1}{L\Delta_s} \leq Resilience_h \leq 1$ or $Resilience_h = -\infty$. In fact, this measure reflects how good the RS responds to a surprise, from its detection at $(h+1)\Pi$ to the recovery at RT_h: 1) It grows through $SurprisePerformance_h$ when the RS is able to safely decide on high performance levels, 2) it grows through $NTTR_h$ when the RS is fast in recovering, e.g., because severity S_h has been low or the harvesting rates are at good conditions and the RS decides appropriately, and 3) it gets $-\infty$ through $SurprisePerformance_h$ when the system is not survivable.

3.1 The RS Objectives

As mentioned before, the RS is either in *normal state* or in *surprise state*. It is usually expected to be in *normal state*, in which it tries to maximize the system performance while preserving its survivability. In case of a *surprise*, it switches to *surprise state* at the boundary of the hyperperiod that the surprise has been recognized, where it behaves differently. Since two consecutive surprises are at least Δ_s hyperperiods apart, the RS should have a time to recovery of at most $\Delta_s\Pi$.

Suppose that the most recent recognized surprise is of hyperperiod h, the most recently completed hyperperiod is h', and that the RS is activated at each hyperperiod boundary. We consider two time intervals: 1) $[(h+1)\Pi, RT_h]$ and 2) $[RT_h, (h'+1)\Pi]$, and the following objectives:

- *Objective 1*: Maximizing $Resilience_h$, as in (8);
- *Objective 2*: Maximizing $NormalPerformance_h$ as in (7).

In the former interval, where the RS is in *surprise state*, it considers when the surprise has happened, how much energy is in the storage unit, and what harvesting rates are predicted, so that it targets Objective 1 to decide on the performance levels of the prediction time horizon, and to apply the performance level of the next hyperperiod. In the latter interval, where the RS is in *normal state*, it considers the earliest time that a surprise is probable, the amount of energy in the storage unit, and the predicted harvesting rates, so that it targets Objective 2 to decide on the performance levels of the prediction time horizon, and to apply the performance level of the next hyperperiod. In Objective 1, maximizing $SurprisePerformance_h$ is in conflict to minimizing $NTTR_h$, because scheduling the tasks at higher performance levels leads to consuming more energy, which in turn increases the time to recovery. Thus, the RS is to consider some trade-off between $SurprisePerformance_h$ and $NTTR_h$. In Objective 2, however, the scheduler is to take care of both performance maximization and the possibility of a surprise with the maximum error of ϵ_{max} in the harvesting rate predictions.

Section 5 proposes the RS with the above mentioned objectives. Before that, in the next section, we describe some useful properties that the proposed RS is based on them.

4 Useful Properties to Design a RS

As mentioned before, $\Delta_s\Pi$, which is the minimum interval between any two consecutive surprises, is considered as the constraint on time to recovery, which should be respected by the RS. This means that for a surprise occurring in hyperperiod h, the time to recovery should be less than $\Delta_s\Pi$, namely we should have $RT_h \leq (h+\Delta_s+1)\Pi$. This necessitates to determine how it can be guaranteed that the RS meets this time constraint. In addition, the system should survive in the presence of such surprises (see Definition 3), i.e., it should be able to plan performance levels of $l \geq 1$ for hyperperiods and it should be able to guarantee that an applied performance level is respected. The RS guarantees such properties by providing some lower bound (for guaranteeing survivability) and some upper bound (for guaranteeing time to recovery) on each element of $\hat{\zeta}(h)$. In the following, we describe some specific properties with regard to survivability and recovery, respectively in Sections 4.1 and 4.2, which lead to have such bounds. Before that, however, we need some properties common to both the survivability and recovery characteristics, as given below.

In the following, we describe the way of storage energy calculation. The amount of energy in the storage unit at the end of an arbitrary hyperperiod is equal to the difference between the total amount of available energy (i.e., the sum of storage energy contents at the start of the hyperperiod and the harvested energy within the hyperperiod) and the amount of consumed energy (by the task set at the planned performance level within the hyperperiod), if the energy storage capacity is unlimited (or it gets not full), namely if there is no wasted energy within the hyperperiod. However, the storage unit capacity is limited to E_{max} in our model; therefore, some energy might be wasted when the storage unit is full and the harvesting rate is greater than the consuming rate. We provide a lemma and a corollary in the following to show that we can use this way of calculation with no concern about its impacts on the system capability for meeting the time to recovery $\Delta_s\Pi$ and the system survivability. Lemma 1 considers the recovery time.

Lemma 1. *Suppose that the RS is in the* surprise state *to recover from the surprise of hyperperiod* h. *If there is some wasted energy within hyperperiod* $h+k$, $1 \leq k \leq \Delta_s$, *then the recovery has been done at the end of the hyperperiod, namely at time* $(h+k+1)\Pi$, *meeting the recovery time constraint of* $(h+\Delta_s+1)\Pi$.

Proof. See Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TC.2022.3202754. □

The following corollary tells that the wasted energy does not negatively impact the system survivability regarding the aforementioned way of storage energy calculation.

Corollary 1. *Suppose that the RS is in* surprise state *to recover from the surprise of hyperperiod* h. *If there is some wasted energy within hyperperiod* $h+k$, $1 \leq k \leq \Delta_s$, *then it makes no problem for the system survivability.*

Corollary 1 is an immediate result of Lemma 1. It tells that if the RS is in *surprise state* and it has planned and guaranteed performance level of $l \geq 1$ using the aforementioned way of

calculation of the storage energy contents, then some wasted energy within a hyperperiod means that the system certainly remains survivable because it recovers in that hyperperiod, namely it faces no energy shortage to run the system according to its plan. Furthermore, because of the recovery, no negative impact of the energy wastage will be spread to the further hyperperiods, so that if the system survivability could be guaranteed in the absence of the surprise, the guarantee is valid after the recovery as well.

More details on the system survivability and recovery are discussed in the following subsections.

4.1 Survivability

Suppose that the system is at the start of hyperperiod h, the initial energy in the storage unit is $E(h\Pi)$, the predicted harvesting rates in the prediction horizon of H hyperperiods are $\hat{R}_h, \ldots, \hat{R}_{h+H-1}$, and the maximum error of the harvesting rate prediction method is ϵ_{max}, which if happens in a hyperperiod, say h, it results in some surprise with severity S_h. To guarantee that the system is survivable, either in the presence or in the absence of a surprise, we are to plan in a manner that the following properties are valid: 1) The system is able to run at least at performance level $l = 1$ within every hyperperiod between $h + 1$ and $h + H - 1$; and 2) the system preserves its pre-planned schedule in hyperperiod h. To this aim, some conditions should be satisfied, as described in the following.

Property 1. *To preserve the survivability of the system in every hyperperiod of the prediction time horizon even if it faces a surprise, some initial energy bound for the hyperperiods are to be respected by the RS.*

To calculate such an initial energy bound for hyperperiod $h + k, 1 \leq k \leq H - 1$, we consider the subset of consuming tasks of the task set Γ at performance level l, denoted by $\Gamma_c^{h+k,l}$, according to the relative power consumption of the tasks at the performance level with respect to the minimum probable harvesting energy rate of the hyperperiod. More formally, we have

$$\Gamma_c^{h+k,l} = \{\tau_i \in \Gamma | pow_{il} > (1 - \epsilon_{max})\hat{R}_{h+k}, 0 \leq k \leq H - 1\}. \quad (9)$$

Using $\Gamma_c^{h+k,l}$, the initial energy bound $\hat{E}((h + k)\Pi)$ satisfying Property 1 can simply be calculated as a sufficient condition for hyperperiod $h + k, 1 \leq k \leq H - 1$, by accumulating the execution time of the consuming tasks at performance level $l = 1$ from the start of the hyperperiod to $\sum_{\tau_i \in \Gamma_c^{h+k}} e_{i1} \frac{\Pi}{\pi_i}$, i.e., to the total execution time of the corresponding jobs within a hyperperiod. Thus, we have

$$\hat{E}((h + k)\Pi) + (1 - \epsilon_{max})\hat{R}_{h+k} \sum_{\tau_i \in \Gamma_c^{h+k,1}} e_{i1} \frac{\Pi}{\pi_i} - \sum_{\tau_i \in \Gamma_c^{h+k,1}} e_{i1} pow_{i1} \frac{\Pi}{\pi_i} \geq 0. \quad (10)$$

According to (10), the total amount of predicted available energy in the time interval of $[(h + k)\Pi, (h + k)\Pi + \sum_{\tau_i \in \Gamma_c^{h+k,1}} e_{i1} \frac{\Pi}{\pi_i}]$ (which is a worst-case scenario, because it fully utilizes the processor during the interval, and it includes all jobs of the energy consuming tasks), i.e., $\hat{E}((h + k)\Pi) + (1 - \epsilon_{max})\hat{R}_{h+k} \sum_{\tau_i \in \Gamma_c^{h+k,1}} e_{i1} \frac{\Pi}{\pi_i}$, is sufficient to schedule all jobs of the consuming tasks at performance level $l = 1$. After some simple algebraic manipulations, (10) provides some safe bound for $\hat{E}((h + k)\Pi), 1 \leq k \leq H - 1$, to be regarded by the RS.

Property 2. *The RS should determine the performance level l of hyperperiod h in a way that success of the schedule be preserved even in the presence of a surprise in that hyperperiod. The following inequality gives a sufficient condition to this aim:*

$$E(h\Pi) + (1 - \epsilon_{max})\hat{R}_h \sum_{\tau_i \in \Gamma_c^{h,l}} e_{il} \frac{\Pi}{\pi_i} - \sum_{\tau_i \in \Gamma_c^{h,l}} e_{il} pow_{il} \frac{\Pi}{\pi_i} \geq 0. \quad (11)$$

According to Properties 1 and 2, two possibilities exist when the system is at time $h\Pi$, which are to be considered together to reach a safe, yet efficient, decision by the RS:

1) No surprise happens in hyperperiod h: Then, $\hat{E}((h + 1)\Pi)$ could be considered as $\hat{E}((h + 1)\Pi) + \epsilon_{max}\hat{R}_h\Pi$ to continue the scheduling for hyperperiod $h + 1$, which may help in guaranteeing its survivability in case of surprise in that hyperperiod.
2) A surprise happens in hyperperiod h: Then, no further surprise will happen until $(h + \Delta_s + 1)\Pi$, so that it is not needed to be conservative with respect to the harvesting rates of the hyperperiods in between, and the RS can decide more relaxed, because the harvesting rates are guaranteed and can be used to preserve survivability. After hyperperiod $h + \Delta_s$, which the recovery has certainly been done and another surprise is possible, the RS takes care of Properties 1 and 2 again.

4.2 Recovery

Suppose that the system is at the start of hyperperiod h, the initial energy in the storage unit is $E(h\Pi)$, the predicted harvesting rates in the prediction horizon of H hyperperiods are $\hat{R}_h, \ldots, \hat{R}_{h+H-1}$, and the maximum error of the harvesting rate prediction method is ϵ_{max}, which if happens in a hyperperiod, say h, it results in some surprise with severity S_h. According to the performance levels planned by the RS, the predicted amounts of energy consumption of hyperperiods over the prediction time horizon are shown as $\hat{EC}_{h+k}$, $0 \leq k \leq H - 1$; these affect the target of recovery $\hat{\zeta}(h)$. The RS should determine the performance levels, and hence the values of $\hat{EC}_{h+k}$, $0 \leq k \leq H - 1$, in a way that if there is a surprise within hyperperiod h, it can perform the recovery in the time interval of $[(h + 1)\Pi, (h + \Delta_s + 1)\Pi]$.

Theorem 1. *Suppose the maximum surprise of hyperperiod h happens, i.e., $R_h = (1 - \epsilon_{max})\hat{R}_h$. Then the RS can recover from the surprise before or at time $(h + \Delta_s + 1)\Pi$ if*

$$\sum_{k=1}^{\Delta_s} \hat{EC}_{h+k} \geq \epsilon_{max}\hat{R}_h\Pi + \Delta_s \sum_{\tau_i \in \Gamma} e_{i1} pow_{i1} \frac{\Pi}{\pi_i}, \quad (12)$$

where $\hat{EC}_{h+k}$ is the amount of predicted energy consumption of hyperperiod $h + k$ based on its determined performance level.

Proof. See Appendix B, available in the online supplemental material. □

The constraint of Theorem 1 implies an upper bound on $\sum_{k=2}^{\Delta_s+1} \hat{E}((h+k)\Pi)$, meaning that the performance levels of the mentioned hyperperiods must be planned in a manner that the energy storage upper bound be distributed among the hyperperiods according to a feasible scenario. If the resulted upper bound for one of the hyperperiods be larger than E_{max}, it means that the recovery is done at the end of that hyperperiod according to the RS plan (see Lemma 1).

If the conditions of Properties 1 and 2, and Theorem 1 are satisfied by the RS, then:

- the RS preserves the system survivability against the possible surprises, because $\hat{\zeta}(h)$ satisfies (10),
- the RS preserves the plan of hyperperiod h against a surprise with the maximum severity with respect to the hyperperiod, because (11) holds, and
- the RS can do the recovery before or at time $(h+\Delta_s+1)\Pi$ if a surprise happens within hyperperiod h with the maximum severity with respect to the hyperperiod, because (12) holds.

5 Resilient Scheduler (RS)

The proposed RS plans for the system performance levels for a prediction time horizon, applies the plan of next hyperperiod, and uses a task scheduler to follow the plan, i.e., the determined performance level, within the hyperperiod. Depending on the RS state, i.e. *normal state* or *surprise state*, it determines the performance levels with different approaches (see Algorithm 1).

In *normal state*, the RS solves a linear programming (LP) with Objective 2 (see Section 3.1) over a time horizon of H hyperperiods with regard to Properties 1 and 2 (i.e., subject to (10) and (11)). In *surprise state*, however, the RS considers some trade-off between the time to recovery and the system performance using a heuristic method towards Objective 1 (see Section 3.1) with regard to Theorem 1 (i.e., subject to (12)).

Normal State. Suppose the system is at the start of hyperperiod h. In *normal state*, the proposed RS solves the LP of (13), assuming that the last surprise has been happened in hyperperiod $h-k'$, $k' \geq 2$ ($k' \geq \Delta_s+1$, if no surprise has been happened before). According to our model described in Section 2, over a time horizon of H hsssssyperperiods, maximizing the energy consumption of the tasks leads to maximizing the $NormalPerformance_h$.

The first constraint in (13) implies that at the start of hyperperiod h, the RS knows the actual amount of energy in the storage unit. The next three constraints demonstrate that there is no surprise within hyperperiods $h-k'+1$ to $h-k'+\Delta_s$, since the last surprise has been occurred in hyperperiod $h-k'$. The fifth constraint considers the energy storage unit capacity. The sixth constraint calculates the amount of energy at the end of each hyperperiod in the prediction time horizon, given the initial energy, the predicted harvesting rates, and the predicted energy consumption of the tasks. The next two constraints represent Properties 1 and 2, respectively. Note that, according to these properties, the system can successfully (with no energy shortage) run the hyperperiod at the selected performance level l. The last constraint considers Theorem 1 to guarantee that in case of surprise in hyperperiod h, the recovery can be done before or at time $(h+\Delta_s+1)\Pi$. Note that if the most recent surprise has been occurred in hyperperiod k', we only can have another surprise if $\Delta_s - k' < 0$; otherwise, the last constraint of (13) has no effect on the solution of LP.

Algorithm 1. Resilient Scheduler (RS)

Input: $E(h\Pi)$, $\hat{R}_h, \ldots, \hat{R}_{h+H-1}$ (Initialization: $k' = \Delta_s$; $SchedulerState = NormalState$; $SurpriseStatePerfs = \emptyset$)
Output: Performance level l_h

$k' = k' + 1$;
if $SchedulerState == NormalState$ *AND* $E(h\Pi) \geq \hat{E}(h\Pi)$ **then**
  Solve the LP of (13);
  $l_h = max\{l \in [1, L] | \sum_{\tau_i \in \Gamma} e_{il} pow_{il} \frac{\Pi}{\pi_i} \leq \hat{EC}_h\}$;
  return l_h;
end
if $SchedulerState == SurpriseState$ *AND* $\zeta(h-k') \geq \hat{\zeta}(h-k')$ **then**
  /* Recovery has been done. */
  Solve the LP of (13);
  $l_h = \max_{1 \leq l \leq L}\{l | \sum_{\tau_i \in \Gamma} e_{il} pow_{il} \frac{\Pi}{\pi_i} \leq \hat{EC}_h\}$;
  $SchedulerState = NormalState$;
  return l_h;
else
  if $SurpriseStatePerfs \neq \emptyset$ **then**
    /* See Lines 19-34. */
    Pick l_h from $SurpriseStatePerfs$;
    return l_h;
  $SchedulerState = SurpriseState$;
  $k' = 1$;
  $SurpriseStatePerfs = \emptyset$;
  S_{h-1} = Severity of the surprise according to (2);
  for $r = \lceil \Delta_s S_{h-1} \rceil$ *to* Δ_s **do**
    Calculate B_r according to (14);
    /* Calc. hyperperiod budgets */
    $Budget_{h+k} = \lfloor \frac{\hat{R}_{h+k}}{\sum_{j=0}^{r-1} \hat{R}_{h+j}} \rfloor B_r, k \in [0, r-1]$;
    /* Calc. performance levels */
    $l_{h+k} = \max_{1 \leq l \leq L}\{l | \sum_{\tau_i \in \Gamma} e_{il} pow_{il} \frac{\Pi}{\pi_i} \leq Budget_{h+k}\}, \quad k \in [0, r-1]$; /* Storage unit initial energy */
    $IE = E(h\Pi)$;
    $feasible = TRUE$;
    for $k = 0$ *to* $r-1$ **do**
      if $IE + \hat{R}_{h+k} \sum_{\tau_i \in \Gamma_c^{h+k, l_{h+k}}} e_{il_{h+k}} \frac{\Pi}{\pi_i} - \sum_{\tau_i \in \Gamma_c^{h+k, l_{h+k}}} e_{il_{h+k}} pow_{il_{h+k}} \frac{\Pi}{\pi_i} \geq 0$ **then**
        $IE += \Pi \hat{R}_{h+k} - \sum_{\tau_i \in \Gamma} e_{il_{h+k}} pow_{il_{h+k}} \frac{\Pi}{\pi_i}$;
      else
        $feasible = FALSE$;
        break;
    if $feasible$ **then**
      $SurpriseStatePerfs = \{l_{h+k} | k \in [1, r-1]\}$;
      return l_h;

After solving (13), the RS finds the maximum performance level l at which the energy consumption of the task set is less than or equal to $\hat{EC}_h$, i.e., $\sum_{\tau_i \in \Gamma} e_{il} pow_{il} \frac{\Pi}{\pi_i} \leq \hat{EC}_h$, and applies it to hyperperiod h; it performs in a similar fashion at the start of the next hyperperiod if no surprise happened in hyperperiod h.

$$\max \sum_{k=0}^{H-1} \hat{EC}_{h+k}$$

$$s.t.\ \ \hat{E}(h\Pi) = E(h\Pi)$$

$$\hat{R}_{h+k} = R_{h+k}, \forall 0 \le k \le \Delta_s - k'$$

$$Rate_{h+k} = R_{h+k}, \forall 0 \le k \le \Delta_s - k'$$

$$Rate_{h+k} = (1-\epsilon_{max})\hat{R}_{h+k}, \quad \forall max((\Delta_s - k'), -1) < k \le H-1$$

$$\hat{E}((h+k)\Pi) \le E_{max}, \ \ \forall 1 \le k \le H$$

$$\hat{E}((h+k+1)\Pi) = \hat{E}((h+k)\Pi) + \hat{R}_{h+k}\Pi - \hat{EC}_{h+k}, \quad \forall 0 \le k \le H-1$$

$$\hat{E}((h+k)\Pi) + Rate_{h+k} \sum_{\tau_i \in \Gamma_c^{h+k,1}} e_{i1}\frac{\Pi}{\pi_i} - \sum_{\tau_i \in \Gamma_c^{h+k,1}} e_{i1} pow_{i1}\frac{\Pi}{\pi_i} \ge 0, \forall 1 \le k \le H-1$$

$$E(h\Pi) + Rate_h \sum_{\tau_i \in \Gamma_c^{h,l}} e_{il}\frac{\Pi}{\pi_i} - \sum_{\tau_i \in \Gamma_c^{h,l}} e_{il} pow_{il}\frac{\Pi}{\pi_i} \ge 0$$

$$\sum_{k=1}^{\Delta_s} \hat{EC}_{h+k} \ge \epsilon_{max}\hat{R}_h\Pi + \Delta_s \sum_{\tau_i \in \Gamma} e_{i1} pow_{i1}\frac{\Pi}{\pi_i}, \ \Delta_s < k'. \quad (13)$$

Although we select performance level l such that $\sum_{\tau_i \in \Gamma} e_{il} pow_{il}\frac{\Pi}{\pi_i} \le \hat{EC}_h$, since $\Gamma_c^{h,l} \subseteq \Gamma$ and we have $E(h\Pi) + Rate_h \sum_{\tau_i \in \Gamma_c^{h,l}} e_{il}\frac{\Pi}{\pi_i} - \sum_{\tau_i \in \Gamma_c^{h,l}} e_{il} pow_{il}\frac{\Pi}{\pi_i} \ge 0$ in (13), Property 2, related to survivability, remains valid.

With regard to Theorem 1, we show through the following theorem that the recovery deadline of $(h+\Delta_s+1)\Pi$ against a surprise in hyperperiod h is also guaranteed.

Theorem 2. *Suppose $\hat{EC}_h$ is determined by (13) and the RS selects the maximum performance level l at which the energy consumption of the task set is less than or equal to $\hat{EC}_h$. If the system energy consumption at level l is less than $\hat{EC}_h$, the guarantee of having the recovery time not later than $(h+\Delta_s+1)\Pi$ in the presence of a surprise in hyperperiod h remains valid.*

Proof. See Appendix C, available in the online supplemental material. □

Surprise State. Suppose we have a surprise in hyperperiod h. Then, the RS switches to *surprise state*, and calculates Δ_s energy consumption budgets $B_1, B_2, \ldots, B_{\Delta_s}$, where B_m, $1 \le m \le \Delta_s$, is the energy budget of the RS to plan the performance levels up to the recovery time if we want to do the recovery at the end of hyperperiod $h+m$. Specifically, if the RS wants to perform the recovery at $(h+m+1)\Pi$, $1 \le m \le \Delta_s$, the following inequality is to be satisfied:

$$E((h+1)\Pi) + \sum_{j=1}^{m} \hat{R}_{h+j}\Pi - B_m \ge \hat{E}((h+m+1)\Pi), \quad (14)$$

where B_m is the energy budget for hyperperiods $h+1$ to $h+m$. Depending on the value of m, $1 \le m \le \Delta_s$, (14) provides different values for $NTTR_h$ and $SurprisePerformance_h$. As it can be seen in Algorithm 1, if there is a surprise in hyperperiod $h-1$, the RS looks for the smallest recovery time between $(h+r)\Pi$ and $(h+\Delta_s)\Pi$ that the task set is feasible according to the calculated budget, where $r = \lceil \Delta_s S_{h-1} \rceil$, $0 < S_{h-1} \le 1$. More precisely, starting with $(h+r)\Pi$, it calculates the energy budget using (14) for the surprise of hyperperiod $h-1$ and distributes the budget among the hyperperiods h to $h+r-1$. Then, it checks the feasibility of the task set, i.e., given the initial energy in the storage unit ($E(h\Pi)$), the next harvesting rates ($\hat{R}_h, \ldots, \hat{R}_{h+r-1}$), and the determined performance levels for hyperperiods h to $h+r-1$,[2] it determines whether the task set can be scheduled without energy shortage in the hyperperiods. If the task set is feasible, then it selects $(h+r)\Pi$ as the recovery time; otherwise, it tries same as above for $(h+r+1)\Pi$. This procedure may continue until $(h+\Delta_s)\Pi$, at which the possibility of recovery is guaranteed based on Theorem 1.

Similar to *normal state*, in *surprise state*, we can discuss the effect of selecting a performance level which consumes less than the allocated energy budget (see Algorithm 1). In fact, doing so will increase the amount of energy at the end of the corresponding hyperperiod, and hence, it will decrease the time to recovery. We give the following corollary.

Corollary 2. *In Algorithm 1, in* surprise state, *suppose $(h+k'')\Pi$ as the planned recovery time, where $r \le k'' \le \Delta_s$. Then, selecting the performance levels for hyperperiods h to $h+k''-1$ with the total energy consumption less than $B_{k''}$ does not violates the recovery time constraint, i.e., we have $RT_{h-1} \le (h+k'')\Pi$.*

We also note that, according to Theorem 1, Algorithm 1 always finds a feasible solution in *surprise state*.

The time complexity of Algorithm 1 depends on the complexity of the algorithm in *surprise state* which is $O(\Delta_s^2\ Ln)$. Since the number of performance levels L and the value of $\Delta_s < H$ are limited and constant in embedded applications, the proposed method has a low time complexity.

6 Simulation Results

In this section, we evaluate the proposed RS using randomly generated task sets as well as four case-studies. We have used Preemptive Fixed Priority as-soon-as-possible (PFP_{ASAP}) [17] as the task scheduler. At each time t, PFP_{ASAP} selects the job of the highest priority active task and executes one time unit of it if there is sufficient energy; otherwise, it leaves the processor idle. We compare the results of the proposed RS to the following methods:

- *Optimal solution (OPT)*: It gives the maximum possible resilience through an exhaustive search among all possible performance levels and time to recoveries over the prediction time horizon. More precisely, it knows the actual and the predicted energy harvesting rates, as well as where the surprise happens; considering the predicted energy harvesting rates before the surprise and the actual energy harvesting rates after the surprise, it calculates all combinations of the possible performance levels, and then it selects the performance levels which maximize the resilience.

2. The performance levels are determined such that the total amount of energy consumption between hyperperiods h and $h+r-1$ is less than or equal to B_r.

- *Conservative Solution (Cons-l1)*: It works similar to the proposed RS, except that it schedules the tasks only at performance level $l = 1$ in *surprise state*.
- *Pure performance maximization algorithm (PERF)*: At each hyperperiod boundary, it solves a LP with the objective of performance maximization over the prediction horizon, with no plan for the surprises.
- *Resilient scheduling with respect to energy changes (RES-EC) [4]*: Assuming that the energy prediction method has no error, i.e., $\epsilon_{max} = 0$, it solves a LP at hyperperiod boundaries with the aim of maximizing resilience against the harvested energy rate changes. RES-EC defines the performance levels according to the (m, k)-firm model [18]. To have comparable results, we set $m = k$ at performance level L in our simulations, and then we generate the other performance levels according to Section 6.1.

6.1 Task Set Generation

We consider task sets of $n = 5$, 10, and 20 tasks, with the hyperperiod limits of 500, 1,000, and 5,000, respectively. The tasks have fixed priorities, where τ_1 has the highest and τ_n has the lowest priorities. The task set utilization is supposed from 0.2 to 1.0, in steps of 0.2. For each n and utilization pair, we generate 100 task sets and average the results over them. The maximum performance level L is set to 5.

The maximum power consumption of each task τ_i, i.e., $pow_{iL}, 1 \leq i \leq n$, as well as the harvesting rates of hyperperiods are generated randomly in the interval of $[1, 100]$, and we consider $E_{max} = 30000$. The power consumption of the tasks at any other performance level $l, 1 \leq l < L$, are generated via multiplying $0.2, 0.4, 0.6$, and 0.8 by pow_{iL}. In the simulations, we assume that $pow_s = 0$. However, we consider some non-zero static power consumption in the case-studies of Section 6.3. We generate the task sets such that they are schedulable by the task scheduler at performance level 1 when there is no energy constraint; thus, we drop the task sets which are not schedulable under such conditions, and then we retry. Further, the simulations are done for different values of ϵ_{max}, S_h, and Δ_s, to represent broad system behaviour; ϵ_{max} can get a value from 0.1 to 0.5, in steps of 0.1; S_h can be from 0.2 to 1.0, in steps of 0.2; and Δ_s can get a value from 5 to 20, in steps of 5. Also, we set $H = 21$ (which is a relatively small value for the prediction time horizon, and further, in all evaluations, $\Delta_s < H$; see Section 2.2). Since having a specific value for S_h, which depends on the values of $\hat{E}((h+1)\Pi)$ and $E((h+1)\Pi)$ (see (2)), is not straightforward in the simulations, we consider 5 consecutive intervals of (0,0.2], (0.2,0.4], (0.4,0.6], (0.6,0.8], and (0.8,1.0], and report the results of each interval for its upper end; e.g., all results of severities in the range of (0,0.2] are reported for $S_h = 0.2$. The simulations are run for 100 hyperperiods.

For each task set, given the hyperperiod limit and the utilization, we generate n periods using the hyperperiod limitation technique [19]. Then, we use UUnifast [20] to generate n utilization values. Next, the maximum execution time of task τ_i, i.e., e_{iL}, $1 \leq i \leq n$ ($L = 5$), is generated via multiplying the utilization and the period of the task. Finally, the execution times of the task at other performance levels l, $1 \leq l < L$, are generated via multiplying $0.2, 0.4, 0.6$, and 0.8 by e_{iL}.

TABLE 1
Failure Ratio of the Considered Methods

	PERF	RES-EC [4]	Cons-l1	RS	OPT
$n = 5$	11.1%	10.8%	1.4%	1.3%	0.9%
$n = 10$	24.5%	25.2%	1.9%	1.8%	1.3%
$n = 20$	46.3%	44.9%	3.4%	3.4%	2.8%

6.2 Performance Evaluation

In this subsection, we first discuss the possibility of failure of our proposed RS with respect to PERF, RES-EC, Cons-l1, and OPT. Depending on the dynamics of energy harvesting and the determined performance levels within a limited prediction time horizon, any of the methods may fail to schedule the task sets due to the possibility of energy shortage sometime within the system lifetime. As it can be seen in Table 1, the failure ratios of PERF and RES-EC are significantly higher than those of the other three methods, because they always work in *normal state*, i.e., they have no intuition of *surprise state*.

In the rest of this subsection, to compare RS to PERF, RES-EC, and Cons-l1, we only consider the task sets which can successfully be scheduled by all these algorithms, i.e., they are feasible with the energy harvesting rates, and thus, they are certainly schedulable by OPT. The simulation results are reported with respect to three criteria: the normal performance in *normal state* (see (7)), the resilience in *surprise state* (see (8)), and the average hyperperiod performance (AHP) of all hyperperiods irrespective of the RS state, where the performance of each hyperperiod is calculated according to (1). All the results are normalized to the corresponding results of OPT ($NormalPerformacne_h^{OPT}$, $Resilience_h^{OPT}$, and AHP^{OPT}, respectively represent the normal performance, resilience, and AHP of OPT).

Figs. 2a, 2b, 2c, and 2d show the normalized AHP for different values of utilization, ϵ_{max}, S_h, and Δ_s, respectively. According to Fig. 2a, by increasing the utilization the normalized AHP decreases. With regard to Algorithm 1, two aspects impact the behavior: the accuracy of feasibility test, and the tightness of lower and upper bounds of the elements of $\hat{\zeta}(h)$. The former gets more accurate by increasing the utilization, however, based on (10), (11), and (12), the lower and upper bounds of $\hat{\zeta}(h)$ get closer to each other by increasing the utilization, which decreases the freedom of the RS in selecting the performance levels for the sake of its conservative behavior. According to Fig. 2a, the conservative behavior overcomes the accuracy of the feasibility test. As shown in Figs. 2b and 2c, by increasing the values of ϵ_{max} and S_h, the normalized AHP increases as well. Because at surprises with higher severity the time to recovery (even for OPT) is usually larger, and thus the RS spends more time in *surprise state* (where it works non-conservative and selects better performance levels); this way AHP depends more on the performance of hyperperiods in *surprise state*, and it behaves more similar to OPT. However, considering PERF and RES-EC, as they have no plan for the surprise, by increasing the values of ϵ_{max} and S_h, the normalized AHP

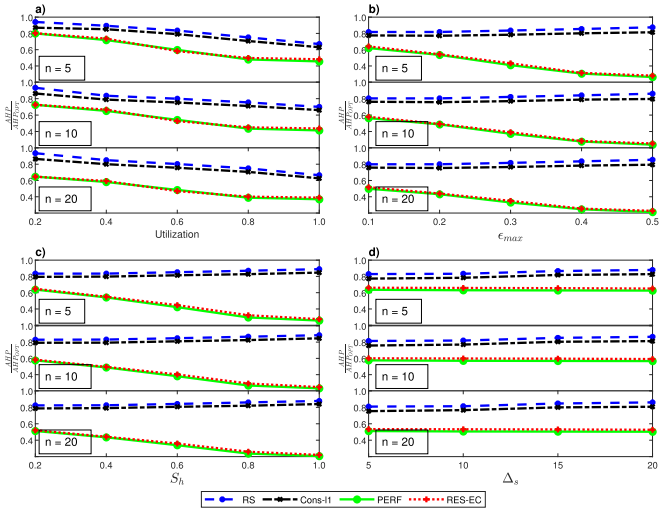

Fig. 2. Normalized AHP.

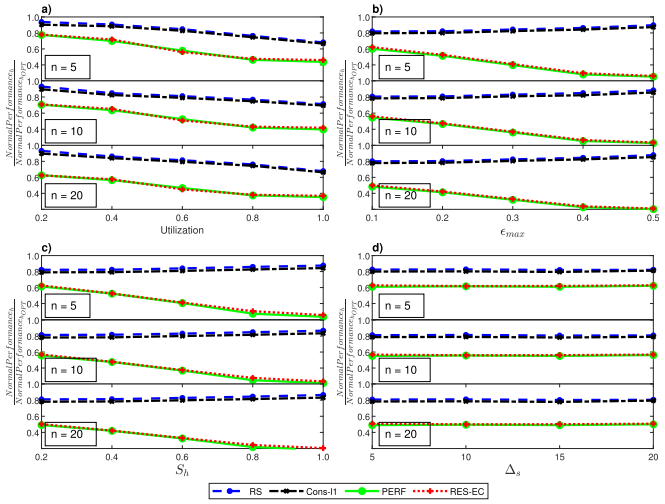

Fig. 3. Normalized $NormalPerformance_h$.

decreases. Further, Cons-l1 has less AHP than RS, because it schedules the tasks only at $l = 1$ in *surprise state*; we note that the amount of wasted energy of Cons-l1 is more than RS. As shown in Fig. 2d, by increasing the value of Δ_s, since the RS has more time for recovery, it can schedule the tasks at higher performance levels, and hence, the normalized AHP slightly increases. Since PERF and RES-EC have no plan for the recovery, increasing the value of Δ_s has no impact on AHP for these algorithms; However, as the AHP of OPT increases, the normalized AHPs of PERF and RES-EC slightly decrease. Δ_s has no significant impact on Cons-l1, because it only schedules the tasks at $l = 1$ in *surprise state*.

Fig. 3 shows the normal performance of the RS, normalized with respect to that of OPT. In *normal state*, the RS conservatively determines the performance levels such that Properties 1, 2, and Theorem 1 hold. This conservatism is more necessary and closer to OPT decisions when ϵ_{max} and S_h increase (see Figs. 3b and 3c). Fig. 3d shows that increasing the value of Δ_s has no significant impact on the normal performance. Regarding PERF and RES-EC, which have no plan against surprise, by increasing utilization, ϵ_{max}, and S_h, the normal performance decreases; and Δ_s has no significant impact on the normal performance of them. Cons-l1 works similar to RS in *normal state*, and it reaches almost the same normal performance. Note that, after the recovery, i.e., at the start of *normal state*, although both RS and Cons-l1 have almost the same energy in the storage unit, their TTRs are different.

In *surprise state*, the objective is maximizing the resilience. Fig. 4 shows the normalized resilience for different values of utilization, ϵ_{max}, S_h, and Δ_s. As it can be seen, by increasing the utilization, the normalized resilience decreases, which is an expected behavior. However, since the RS keeps some trade-off between the surprise performance and time to recovery considering the severity of the surprises (see Algorithm 1), the normalized resilience does not change significantly for different values of ϵ_{max} and S_h, which is an important property. By increasing the value of Δ_s, the RS selects the performance levels at *surprise state* less conservatively. This increases the value of $NTTR_h$ and $SurprisePerformacne_h$, affecting the resilience (see (8)); Fig. 4d shows this effect is minor irrespective of the lower time complexity of RS than OPT, i.e., another important property of the former. Comparing to PERF, RES-EC, and Cons-l1, the resilience of RS outperforms these three algorithms is all the cases.

According to Figs. 2, 3, and 4, the results for different values of n are close to each other, implying the scalability of the proposed RS. Regarding its time complexity, on a Core i5 3.3 GHz CPU with 8 GB of memory, the average execution times for $n = 5$, 10, and 20, are respectively 0.12, 0.21, and 0.43 ms for a hyperperiod.

6.3 Case-Study

This section shows the applicability and effectiveness of the proposed RS through four case-studies: 1) Vehicle monitoring [21], 2) patient monitoring [22], 3) forest monitoring [23], and 4) computational tasks [24]. The first three case-studies are on WINS 1, MSP430, and MTS400 boards, respectively, and all of them include a multi-sensor board, some attached sensors, a battery, and an antenna to send the information to a base station. The last one uses a LEON3 processor, as a common processor in embedded systems like satellites.

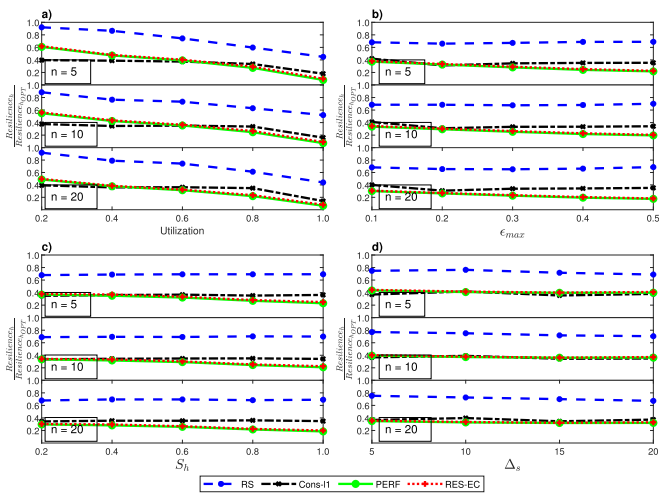

Fig. 4. Normalized $Resilience_h$.

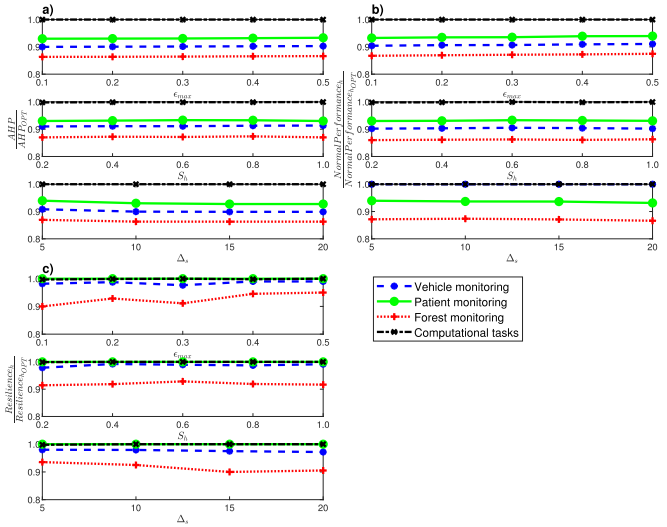

Fig. 5. Normalized AHP, $NormalPerformance_h$, and $Resilience_h$.

The vehicle monitoring has five tasks (accelerometer, acceleration, acoustic, motion detection, and temperature), the patient monitoring has six tasks (accelerometer, heart rate/ECG, acoustic, temperature, humidity, and low power GPS), and the forest monitoring has eight tasks (wind speed, acoustic, motion detection, magnetic, temperature, soil moisture, humidity, and low-power image). The tasks of the computational case-study are from MiBench Benchmark suite [24] which includes 32-bit CRC, QSORT, and ISPELL (a fast spelling checker). The specification of the task set of these case-studies are taken from [5].[3]

In addition to the above-mentioned tasks, each case-study includes the proposed RS as a separate task of the highest priority with a period of the hyperperiod length. Regarding the execution time of the RS task, we first obtain the execution time of the proposed resilient scheduler on a Core i5 3.3 GHz CPU with 8 GB of memory. Then, for each case-study and according to the corresponding processor speed, we estimate the execution time of the RS task. For vehicle monitoring, patient monitoring, forest monitoring, and computational tasks on a Core i5 3.3 GHz CPU with 8 GB of memory, we respectively obtained 0.09, 0.1, 0.12, and 0.06 ms as the average RS execution times for one hyperperiod. On the other hand, the CPU frequency for their hardware are 133 MHz, 16 MHz, 16 MHz, and 400 MHz, respectively. Therefore, we estimate the execution time of the proposed resilient scheduler as 2.24, 20.625, 24.75, and 0.495 ms, respectively for them. Regarding the power consumption of the RS task, for each case-study, we consider the maximum power consumption of the processor as the power consumption of the proposed RS. Therefore, we set the power consumption of the RS task to 330 [25], 8.096 [26], 2,400 [27], and 1,500 mW [24] for vehicle monitoring, patient monitoring, forest monitoring, and computational tasks, respectively.

The energy harvesting rates of April 16, 2022 are taken from Solar Radiation Lab (SRL) [28], assuming a solar panel of size 40 cm^2. The granularity of the solar data is 1 minute. The hyperperiod length in the mentioned case-studies is 2 minutes. Regarding that the rate of harvesting energy in the data is the same for two consecutive hyperperiods in most cases, we conservatively consider the minimum harvesting rate of the two consecutive minutes as the harvesting rate of the related hyperperiod. The static power consumption of the case-studies according to their hardware platforms (see [5]) are 40 mW, 0.506 mW, 310 mW, and 43.11 mW for vehicle monitoring, patient monitoring, forest monitoring, and computational tasks, respectively.

The values of ϵ_{max} and Δ_s depend on the energy source (e.g., solar, wind, etc.) and the energy prediction method accuracy. Since we do not consider any specific energy prediction method, same as Section 6.2, we report the results of the case-studies with respect to different values of ϵ_{max} and Δ_s. Also, since the value of S_h depends on the amount of energy in the storage unit at the hyperperiod boundaries (see (2)), the results are reported for different values of S_h. Fig. 5 shows the normalized AHP, $NormalPerformance_h$, and $Resilience_h$ of the proposed RS w.r.t. OPT. As can be seen, the proposed RS can achieve effective performance and resilience on the case-studies for different values of ϵ_{max}, S_h, and Δ_s. Further, the results show that the values of ϵ_{max}, S_h, and Δ_s have no significant impact on the performance/resilience of the proposed scheduler, implying the applicability of the proposed resilient scheduler. Matlab source code for the proposed resilient scheduler that utilizes the YALMIP toolbox [29] is available on https://github.com/mshirazi1189/ResilientScheduler.

7 Conclusion

In this paper, we consider the problem of resilient scheduling against the energy harvesting rate prediction error. We provide some theoretical aspects to show how the energy-resilient scheduler can be survivable, and do the recovery within some time constraint. We also propose a measure of resilience which composes performance, survivability, and time to recovery together. The simulation results show that the proposed energy-resilient scheduler effectively approximates the optimal solutions, and it reacts appropriately against surprises of high severity.

References

[1] J. Simpson and E. Weiner, *The Oxford English Dictionary*, 2nd ed., Oxford, U.K.: Clarendon Press, 1989.
[2] M. Bishop, M. Carvalho, R. Ford, and L. M. Mayron, "Resilience is more than availability," in *Proc. New Secur. Paradigms Workshop*, 2011, pp. 95–104.
[3] J. F. Meyer, "Model-based evaluation of system resilience," in *Proc. Annu. IEEE/IFIP Conf. Dependable Syst. Netw.*, 2013, pp. 24–27.
[4] M. Shirazi, M. Kargahi, and L. Thiele, "Resilient scheduling of energy-variable weakly-hard real-time systems," in *Proc. 25th Int. Conf. Real-Time Netw. Syst.*, 2017, pp. 297–306.
[5] M. Shirazi, M. Kargahi, and L. Thiele, "Performance maximization of energy-variable self-powered (m, k)-firm real-time systems," *Real-Time Syst.*, vol. 56, pp. 64–111, 2020.
[6] N. Stricker and L. Thiele, "Analysing and improving robustness of predictive energy harvesting systems," in *Proc. 8th Int. Workshop Energy Harvesting Energy-Neutral Sens.*, 2020, pp. 64–70.
[7] A. Bhat, S. Samii, and R. Rajkumar, "Recovery time considerations in real-time systems employing software fault tolerance," in *Proc. 30th Euromicro Conf. Real-Time Syst.*, 2018, pp. 1–22.

3. The values of pow_{il} and $e_{il}, 1 \leq i < L, L = 5$, are set via multiplying 0.2, 0.4, 0.6, and 0.8 by pow_{iL} and e_{iL}, respectively.

[8] P. P. Nair, A. Sarkar, and S. Biswas, "Fault-tolerant real-time fair scheduling on multiprocessor systems with cold-standby," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 4, pp. 1718–1732, Jul./Aug. 2021.
[9] J. Saber-Latibari, M. Ansari, P. Gohari-Nazari, S. Yari-Karin, A. M. H. Monazzah, and A. Ejlali, "Ready: Reliability- and deadline aware power-budgeting for heterogeneous multicore systems," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 40, no. 4, pp. 646–654, Apr. 2021.
[10] F. Baharvand and S. G. Miremadi, "LEXACT: Low energy n-modular redundancy using approximate computing for real-time multicore processors," *IEEE Trans. Emerg. Topics Comput.*, vol. 8, no. 2, pp. 431–441, Apr.–Jun. 2020.
[11] Q. Han, M. Fan, L. Niu, and G. Quan, "Energy minimization for fault tolerant scheduling of periodic fixed-priority applications on multiprocessor platforms," in *Proc. IEEE Des., Autom. Test Europe Conf. Exhib.*, 2015, pp. 830–835.
[12] Y. Ma, T. Chantem, R. P. Dick, and X. S. Hu, "Improving reliability for real-time systems through dynamic recovery," in *Proc. IEEE Des., Autom. Test Europe Conf. Exhib.*, 2018, pp. 515–520.
[13] H. Sobhani, S. Safari, J. Saber-Latibari, and S. Hessabi, "REALISM: Reliability-aware energy management in multi-level mixed-criticality systems with service level degradation," *J. Syst. Archit.*, vol. 117, 2020, Art. no. 102090.
[14] I. Ali, Y.-I. Jo, S. Lee, W. Y. Lee, and K. H. Kim, "Reducing dynamic power consumption in mixed-critical real-time systems," *Appl. Sci.*, vol. 10, no. 20, 2020, Art. no. 7256.
[15] K. Cao, G. Xu, J. Zhou, M. Chen, T. Wei, and K. Li, "Lifetime-aware real-time task scheduling on fault-tolerant mixed-criticality embedded systems," *Future Gener. Comput. Syst.*, vol. 100, pp. 165–175, 2019.
[16] A. Haeberlin et al., "The first batteryless, solar-powered cardiac pacemaker," *Heart Rhythm*, vol. 12, no. 6, pp. 1317–1323, 2015.
[17] Y. Abdeddaïm, Y. Chandarli, and D. Masson, "The optimality of PFP_{ASAP} algorithm for fixed-priority energy-harvesting real-time systems," in *Proc. IEEE 25th Euromicro Conf. Real-Time Syst.*, 2013, pp. 47–56.
[18] M. Hamdaoui and P. Ramanathan, "A dynamic priority assignment technique for streams with (m, k)-firm deadlines," *IEEE Trans. Comput.*, vol. 44, no. 12, pp. 1443–1451, Dec. 1995.
[19] Y. Abdeddaïm, Y. Chandarli, and D. Masson, "Limitation of the hyper-period in real-time periodic task set generation," in *Proc. RTS Embedded Syst.*, 2001, pp. 133–147.
[20] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Syst.*, vol. 30, pp. 129–154, 2015.
[21] H. Marcy, J. Agre, C. Chien, L. Clare, N. Romanov, and A. Twarowski, "Wireless sensor networks for area monitoring and integrated vehicle health management applications," in *Proc. IAA Guidance, Navigation, Control Conf. Exhib.*, 1999, pp. 1–11.
[22] B. P. Lo, S. Thiemjarus, R. King, and G.-Z. Yang, "Body sensor network - A wireless sensor platform for pervasive healthcare monitoring," in *Proc. 3rd Int. Conf. Pervasive Comput.*, 2005, pp. 77–80.
[23] Y. Li, Z. Wang, and Y. Song, "Wireless sensor network design for wildfire monitoring," in *Proc. IEEE 6th World Congr. Intell. Control Automat.*, 2006, pp. 109–113.
[24] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *Proc. IEEE Int. Workshop Workload Characterization*, 2001, pp. 3–14.
[25] Intel strongarm sa-1100 microprocessor. Aug. 1999. [Online]. Available: https://courses.cs.washington.edu/courses/cse466/00au/sa1100dev.pdf
[26] Msp430 datasheet. May 2013. [Online]. Available: https://www.ti.com/lit/ds/symlink/msp430g2553.pdf
[27] Atmega128 1 datasheet. Jun. 2011. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/doc2467.pdf
[28] J. Ramos and A. Andreas, "University of texas panamerican (UTPA): Solar radiation lab (SRL), Edinburg, Texas (data)," National Renewable Energy Lab. (NREL), Golden, CO, USA, Rep. no. DA-5500–56514, 2011.
[29] J. Lofberg, "YALMIP: A toolbox for modeling and optimization in MATLAB," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2004, pp. 284–289.

Mahmoud Shirazi received the PhD degree in computer science from the Institute for Research in Fundamental Sciences (IPM), Tehran, Iran, in 2019. He is currently an assistant professor with the Department of Computer Science and Information Technology, Institute for Advanced Studies in Basic Sciences (IASBS). His research interests include energy aware/efficient scheduling of real-time systems and high performance embedded computing.

Lothar Thiele received the Dr-Ing degree in electrical engineering from the Technical University of Munich, in 1985. He joined ETH Zurich, Switzerland, as a full professor of computer engineering, in 1994, where he currently leads the Computer Engineering and Networks Laboratory. His research interests include models, methods and software tools for the design of embedded systems, embedded software and bioinspired optimization techniques.

Mehdi Kargahi (Senior Member, IEEE) received the PhD degree from the Sharif University of Technology, in 2006. He is currently an associate professor with the School of Electrical and Computer Engineering, University of Tehran. He is currently leading the Dependable/ Distributed Real-Time Systems Research Lab (http://drts.ut.ac.ir). His research interests include analysis and management of energy and performance of cyber-physical systems.