PEDASA: Priority, Energy and Deadline Aware Scheduling Algorithm

Maroua Gasmi[1,2](✉), Olfa Mosbahi[2], Mohamed Khalgui[2], and Luis Gomes[3]

[1] Faculty of Sciences of Tunis, University Tunis El Manar, Tunis, Tunisia
mra.gsm@gmail.com
[2] LISI Lab, INSAT Institute, University of Carthage, Tunis, Tunisia
{olfamosbahi,khalgui.mohamed}@gmail.com
[3] Universidade Nova de Lisboa, Lisbon, Portugal
lugo@fct.unl.pt

Abstract. We present a new approach for scheduling workloads containing periodic tasks in soft real-time systems. The proposed algorithm consists on finding a new set of priorities depending of the three main criteria identified in a real-time system: fixed priority initially assumed by user, deadline and energy efficiency. Our proposition involves a computational procedure that is responsible of extracting the new values of priorities out of the importance of the three factors previously mentioned. An eventual re-adjustment of the deadlines is also faced all along with the reloading of the system's power on specified instants. The resulting system is, therefore, feasible and effectively schedulable compared to the mono-criteria algorithms. This contribution allows also the definition of precise instants of reloading which enforces the new concept of extending the lifetime of the system.

Keywords: Real-time · Multi-criteria · Scheduling and optimization · Energy efficiency

1 Introduction

In a world where technology does not stop evolving, basic daily activities are substituted by extremely intelligent systems that keep getting optimized by time. Real-time systems offer a big range of services adapted. Moreover, these systems consist of one or more subsystems that should respond in a precise and finite time specified by the external world. Thus, a result obtained after a stated deadline remains false even if it is logically right. Consequently, the total correctness of an operation depends not only upon its logical correctness, but also upon the time in which it is performed which depends on several criteria. Although many scheduling algorithms concentrate only on timing constraints, others exist and need to be acknowledged as well. For this purpose, some works got oriented to the scheduling using a multi-criteria method. Nevertheless, it is more realistic to find compromises between the different parameters than to choose a single parameter at once. This way, it is possible to partially satisfy the varied objectives.

The basic problem in this paper is how to deal with scheduling the existing tasks while keeping an eye on all of the criteria at once. In the case of a soft real-time system, it seems crucial to watch the feasibility of most of the tasks while calculating the new priorities. As an example of works who got concerned about this matter, the author in [11] introduced a multi-criteria algorithm to schedule soft real-time tasks on uniform multiprocessor systems. This approach uses three criteria, namely deadline, laxity and interval. Similarly, In the work [2], the approach is carried considering the priority, the processing time as well as the waiting time of a task. The previously mentioned work, as well as the approach introduced in [10] treat several parameters while scheduling real-time tasks by considering a fuzzy reasoning. Although this reasoning is capable of dealing with uncertainties in a many-valued logic, there are more than the simple "true" or "false" responses. In fact, this logic only disposes of approximations rather than fixed and exact analysis especially when extreme precision is required. Another limiting factor of fuzzy reasoning is the extensive testing it needs to find an adequate decision. In the contrary, the algorithm RT-DBP [1] takes into account many scheduling parameters. All the criteria are associated with weight parameters in order to give more weight to one criterion in particular depending on the application requirements. Although this approach uses computational procedures instead of estimations, the calculations are made during run-time which can have an impact on the global time processing. Like all the mentioned works, our approach treats the setting of new priorities depending on the existing criteria. It computes the latter mentioned priorities and finds the exact moments where the reloading of the system energy should take place. As a matter of fact, the particularity of the work is that it does not totally ignore the predefined parameters. Alternatively, it relates to them when it comes to calculating the new ones. Among these parameters we specifically find the first set up priority which indicates the degree of importance that a prospective user accords to the task. The other parameters are eventually, the deadline which is a crucial real-time efficiency indicator and the level of energy consumption which has a great impact on the functioning of the system in general. Another advantage presented in our proposition is the calculation that is performed in an off-line mode. This way, no computational overheads are imposed on the system during its execution. In order to explicit our proposition, we expose in the following section the miscellaneous parameters defining a real-time system model as well as the most substantial static scheduling algorithms dealing, separately, with the priority, deadline and energy consumption parameters. The impact of applying each of these algorithms on a predefined set of tasks is pinpointed through Sect. 3. After mathematically formalizing the needed elements in Sect. 4, we detail the proposed solution PEDASA in the section that follows. We first start by analyzing the potential priorities carried out of the predefined parameters. The exact values of these priorities are, then, set by checking the possibility of a maximum regard to deadline in parallel with the consideration of the initial priorities respect all along with an adjustment of periods when necessary. Counting on the resulting set of priorities, we proceed to the search of the exact moments at which the energy level in a system can be reloaded. This enlarges the time span of the tasks and gives them the possibility to execute without any power constraints imposed on them.

2 Background

In this section we aim to introduce the basic axes on which stands the context of this work. Therefore, we conduct a definition of the elemental real-time model as well as an overview on the existing mono-criteria scheduling algorithms that allow a more effective arrangement of the latter.

2.1 Real-Time System Model

Concretely, a real-time system should necessarily guarantee a response within strict time constraints, referred to as deadlines. For this matter, three classes are introduced: **(i)** *Hard* where missing a deadline is totally fatal, **(ii)** *Firm* where few deadline misses are allowable, but may affect the quality of service within the system and **(iii)** *Soft*, like our chosen system, where the practicality of a result debases after its deadline, thereby altering the efficiency of the system [12]. In a multi-tasking method, several tasks can be part-way through execution at the same time, and more than one task is advancing over a specific period of time. The parameters of a real-time task, denoted as τ_i, are mainly: **(i)** P_i Static priority representing the degree of functional importance related to the task, **(ii)** $Order_i$ Order of execution of a task, **(iii)** A_i Arrival time, **(iv)** S_i Actual starting time, **(v)** C_i Computation time also known as Worst Case Execution Time (WCET), **(vi)** T_i Period, **(vii)** D_i Deadline, **(viii)** R_i Response time, **(ix)** F_i Actual time at which the task finishes its execution, **(x)** BF_i Best finishing time and **(xi)** E_i Percentage of power needed for each task. The execution of the periodic tasks within a system is repeated every hyper-period, denoted by T_{hyp}. The latter is introduced as the smallest interval of time after which the periodic patterns of all the tasks are repeated. It is typically defined as the LCM (least common multiple) of all the periods. In this paper, we are interested in computing a new set of priorities in order to maximize the number of feasible tasks.

2.2 Mono-criteria Scheduling Algorithms

A scheduling algorithm enables the orchestration of priorities to the set of tasks. The performance of such an algorithm is judged for how quickly or how predictably a scheduled real-time system can respond [8]. Therefore, the assigned priorities are based on deadline or some other timing constraint. In a time shared scheduling, such in a multi-tasking system, a scheduler has the power to preempt a task and to resume its execution after a while [9]. This change is known as a context switch and several scheduling algorithms use preemption in order to finish the execution of a higher priority task. In literature, two classes of real-time scheduling algorithms are introduced: static or dynamic [5]. In static scheduling, decisions are made during compilation. The parameters of all the tasks are assumed to be known in advance and a schedule is built based on this assumption. Consequently, no modification can be applied online once the scheduler is set. On the other hand, the decisions related to the dynamic scheduling are done at run-time. Although, dynamic schedulers are flexible and adaptive, they can cause a significant overheads because of consuming run-time processing. Reason why, several industrial propositions insist on making use of the static scheduling algorithms instead. Usually, these algorithms use a single criteria for determining the priorities of the different tasks. This priority can be fixed ahead, as in Fixed Priority Preemptive (FPP) algorithm, or concluded out of another parameter. By parameter we can refer to the period of the task, as in Rate Monotonic Scheduling (RM), or energy consumption as in Low Energy First (LEF). In the Fixed Priority Preemptive Algorithm (FPP), the scheduler makes sure that at any instant, the processor executes the highest priority task among the rest of the tasks that are currently waiting for execution [13]. As for the Rate Monotonic scheduling algorithm (RM), the static priorities are assigned based on task periods [6]. The task with the shortest period gets the highest priority, and the one whose the period is the longest gets the lowest static priority. Since in most cases, the period of the task is analogous with its deadline, the Deadline Monotonic algorithm is an extension of RM. Considering the energy consumption parameter, several works treated the case where the lower the power consumption of a task the most prioritized it is [7]. Compared to the mentioned algorithms, PEDASA introduces the possibility of assigning new priorities that take into consideration the three criteria all at once. It also offers the ability of defining a set of moments at which the energy reloads enhancing therefore the efficiency of the complete system.

3 Case Study

Through this section, we show by a concrete example how choosing a single scheduling criteria does not allow the satisfaction of others. For this matter, let us take an example of a set of six tasks. As mentioned in Sect. 2, each one is characterized by a static priority P, a period T that also corresponds to its deadline, a percentage of energy consumption E and an execution time C. The different values of these criteria are given by Table 1. It is to mention that the

Table 1. Parameters of the tasks.

	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6
P	6	5	4	3	2	1
T	120	170	50	80	110	100
E	60	30	50	10	40	20
C	30	10	10	20	10	10

release time of all the tasks is 0 (e.g. all tasks are assumed to be synchronous). Each time, we use a scheduling algorithm based on one of the assumed criteria. At a first time, Fixed Priority Protocol (FPP) is applied to the set of tasks then Rate Monotonic (RM) and finally the Low Energy First Protocol (LEF).

3.1 Fixed Priority Protocol (FPP)

When using the FPP algorithm, the relation between the priorities of the different tasks is defined by the Eq. 1:

$$P_1 > P_2 > P_3 > P_4 > P_5 > P_6 \tag{1}$$

The values of the response times using this protocol are $\{R_1 = 30; R_2 = 40; R_3 = 50; R_4 = 80; R_5 = 120; R_6 = 220\}$. Although the tasks conserve their fixed priorities, it is obvious that some of them miss their deadlines. In fact, only the execution of the tasks τ_1, τ_2, τ_3 and τ_4 goes normally. As for the tasks τ_5 and τ_6, their response times exceed their deadlines. Considering the energy consumption when using this scheduling algorithm, τ_1 and τ_2 are the only ones executing since they consume 80 % of the battery.

3.2 Rate Monotonic (RM)

Since the scheduling of the tasks depends on their periods, the relation between the priorities of these tasks is given by the Eq. 2. This way none of the following tasks respects the initially given priorities.

$$P_3 > P_4 > P_6 > P_5 > P_1 > P_2 \tag{2}$$

Therefore, using this scheduling algorithm only τ_3, τ_4, τ_5 and τ_6 (the tasks with the highest priorities) respect their deadlines. Therefore, the values of the response time are $\{R_1 = 140; R_2 = 240; R_3 = 10; R_4 = 30; R_5 = 50; R_6 = 40\}$. Changing the order of execution allows τ_3, τ_4 and τ_6 to operate with the existing power.

3.3 Low Energy First Protocol

When using the Energy aware algorithm, the relation between the priorities of execution of the different tasks is defined by the Eq. 3 which is totally different from the initial one.

$$P_4 > P_6 > P_2 > P_5 > P_3 > P_1 \tag{3}$$

The execution of the tasks proves that the tasks τ_3 and τ_1 do not respect their deadlines. The corresponding response times are $\{R_1 = 160; R_2 = 40; R_3 = 60; R_4 = 20; R_5 = 50; R_6 = 30\}$. For a single criteria scheduler, it is difficult to concentrate on all the different parameters of a real-time system. In the works related to these scheduling algorithms, the focus on one criteria at a time leads to ignoring the other ones. This been said, the originality of the proposed approach lies in respecting all of the mentioned parameters in order to get a scheduled system that is aware of the deadline constraints and the energy effectiveness.

4 Formalization

Through this section, we tend to mathematically represent the underlying assumptions. Moreover, we expose the problem treated in this paper analytically. We also formalize all the aspects related to energy, deadline and initially fixed priorities in order to have a precise overview on the given problem.

4.1 Energy

The studied system *Sys* is composed of n tasks $\{\tau_1 ... \tau_n\}$ and takes into account the energy consumption aspect. We suppose that obtaining the information about the energy does not require any power consumption and that the battery level gets restored every specific amount of time. As a consequence, let E(t) be the level of available energy at the moment t. For a non preemptive scheduling algorithm, it is possible to know the amount of energy consumed at a specific moment. The function allowing such awareness is denoted by $E_i^{'}(t)$ and given by Eq. 4. Therefore, a precise task τ_i (i = 1..n) does not consume any power before its starting time S_i and right after its best finishing time BF_i (at this level, we consider a non preemptive task). However inside of the interval limited by these two values, the energy consumption increases continuously following a general function *Consump*.

$$E_i^{'}(t) = \begin{cases} 0 \; if \; t \leq S_i \vee t > BF_i \\ E_i \; if \; t = BF_i \\ Consump(t, BF_i, S_i) \; otherwise \end{cases} \tag{4}$$

where *Consump* represents the exact equation related to the consumption. We assume that we do not really have a precise idea on the latter mentioned function. That's why in this paper we are supposing that the energy is consumed exponentially. The amount consumed at the starting moment is 0 and at the best finishing time the whole amount related to the task is drained.Therefore, we propose a function that allows having the previously mentioned characteristics and is given by Formula 5. The scope of this function denoted by β allows having a proportional increase of consumption and is calculated through the couple of values of *Consump* obtained at S_i and BF_i.

$$\begin{gathered} Consump(t, BF_i, S_i) = 1 - \exp^{-\beta(t-S_i)} \\ \beta = \frac{-log(1-\frac{E_i}{100})}{BF_i - S_i} \end{gathered} \tag{5}$$

The power consumption in the system is also dependent on the priority of the tasks. In fact, the energy is consumed starting from the more prioritized task and going on. In the best cases, all of the n tasks are executed before the total exhaustion of the battery. This means that if the energy is consumed in a specific order, it is preferable that the sum of the amount of energy consumed does not exceed 100 % of the available power. This case is represented by Eq. 6.

$$\sum_{i=1}^{n} EP(i)/100 < 1 \tag{6}$$

where EP(i), given by Eq. 7, represents the percentage of energy consumed by the task that the value of its priority is $n+1-i$ and, as a consequence, its range of execution is i.

$$EP(i) = \{E_j / \exists j, 0 < j \leq n \wedge P_j = n + 1 - i\} \tag{7}$$

In the general case, let $T_{ES}(SA)$ be the set of tasks that are executed before the total exhaustion of the battery. The number of elements in this set, given by Eq. 8, totally depends on the chosen scheduling algorithm SA.

$$T_{ES}(SA) = \{\exists \tau_i / 0 < i \leq n, \sum_{i=1}^{n} EP(i)/100 < 1\} \tag{8}$$

In the best case the number of the elements of $T_{ES}(SA)$ corresponds to the total number of tasks in the system. This can be translated by the fact that all the tasks are applied without the total tiredness of the energy. This is described by Eq. 9.

$$Card(T_{ES}(SA)) = Card(T_s) = n \tag{9}$$

4.2 Deadline

The respect of deadline within a real-time system is also a crucial need that demands attention. Therefore, the best real-time scheduling algorithm is the one that allows all the tasks to finish their execution before reaching their deadlines. Let $T_{DS}(SA)$ be the set of tasks that respect their deadlines under a given scheduling algorithm SA. This set is represented by Eq. 10 where the elements are only the tasks whose their execution (the sum of the arriving and the response times) does not exceed the predefined deadline.

$$T_{DS}(SA) = \{\forall i, \exists \tau_i / 0 < i \leq n \wedge A_i + R_i \leq D_i\} \tag{10}$$

The whole system is considered feasible when the number of elements in the set $T_{DS}(SA)$ is equal to n (the number of tasks in the set T_s). This is represented by Eq. 11.

$$Card(T_{DS}(SA)) = Card(T_s) = n \tag{11}$$

4.3 Priority

The priorities initially fixed in the system are sometimes of extreme functional importance. Thus, ignoring them for the profit of other parameters can affect the whole system in an undesirable way. In the perspective of this criterion, let $T_{PS}(SA)$ be the set of tasks that conserve their initial priorities under a given scheduling algorithm SA. This set is represented by Eq. 12. We suppose that IR_i is the initial range corresponding to a task τ_i and resulting from the initial given priority. $FR_i(SA)$, on the other hand, corresponds to the final range obtained after applying a feasible scheduling algorithm SA. These values correspond to the order in which the task should be executed. The following set contains only the tasks that conserved their order (the ones where $IR_i = FR_i$).

$$T_{PS}(SA) = \{\forall i, \exists \tau_i / 0 < i \leq n \wedge IR_i = FR_i(SA)\} \tag{12}$$

Similarly, the system is completely feasible if the number of elements in the set $T_{PS}(SA)$ is equal to n (the number of tasks in the set T_s). This is represented by Eq. 13.

$$Card(T_{PS}(SA)) = Card(T_s) = n \tag{13}$$

4.4 Generalization

Generally speaking, applying a scheduling algorithm can affect the system in 3 different ways. In the first possibility (or the best case), the Eqs. 9, 11 and 13 are all perfectly verified. In the second case, only one of the two conditions is met and in the worst case none of them is verified. Let ES, DS and PS be respectively the Boolean verification functions of the Eqs. 9, 11 and 13. Therefore the formalization of the states that a system can have is given by formula 14.

$$\forall \tau_i \in T_s, State(Sys) : \begin{cases} ES \wedge DS \wedge PS(\textbf{State 1}) \\ \vee \\ \neg ES \wedge DS \wedge PS(\textbf{State 2}) \\ \vee \\ \neg ES \wedge \neg DS \wedge PS(\textbf{State 2}) \\ \vee \\ ES \wedge \neg DS \wedge \neg PS(\textbf{State 2}) \\ \vee \\ ES \wedge DS \wedge \neg PS(\textbf{State 2}) \\ \vee \\ \neg ES \wedge \neg DS \wedge \neg PS(\textbf{State 3}) \end{cases} \tag{14}$$

The pinpointed problem is basically how to get the priority, fixed by a user depending on its preferences, to be respected all along with the deadline, which is a crucial real-time parameter, and the energy efficiency. The following section details the proposed solution that solves this confusion in a simple and effective way.

5 Proposed Approach: PEDASA

The solution to the problem mentioned in Sect. 4 is given by the proposed scheduling algorithm entitled PEDASA. It is a static algorithm that consists on manipulating certain parameters in order to allow all of tasks to meet their deadlines and to execute before the exhaustion of the battery. The new suggested priorities take into consideration the importance of the 3 principle criteria: **(i)** fixed priority, **(ii)** deadline and **(iii)** energy efficiency in the perspective of each task.

5.1 Real-Time Reasoning

New Priorities Analysis. The procedure of computing the new priorities is based on the three importance factors previously mentioned. In fact, α_P, α_D and α_E are rates related to the importance of respectively the existing priority, the fixed deadline and the energy efficiency. The relation between these rates is given by Eq. 15.

$$0 \leq \alpha_P, \alpha_D, \alpha_E \leq 1, \alpha_P + \alpha_D + \alpha_E = 1 \tag{15}$$

The next step consists on sorting the tasks, each time depending on the parameter, while according a positive number that corresponds to their order. Let m_{ij} be the order factor; *i (1..n)* corresponds to the index of the task and j to the parameter (1: Fixed Priority, 2: Deadline and 3: Energy consumption). This number does not exceed the number of elements in the task set T_s. When sorting the tasks, we consider an ascending order for the fixed priority and a descendant one for the deadline and the energy consumption. Let M be the matrix relating the order factors of the tasks to the defined parameters. This matrix, having as dimensions n (the number of tasks) rows and 3 (corresponds to the system parameters) columns, is composed of the elements m_{ij}. Let K(τ_i) be the value of the new priority of the task τ_i that depends on the importance factors α_P, α_D and α_E, and the order factors of this latter. We suppose that K is a vector that contains the new priorities of all the tasks. Therefore, Eq. 16 defines the value of these new priorities.

$$K = \begin{pmatrix} \alpha_P \\ \alpha_D \\ \alpha_E \end{pmatrix} * M \tag{16}$$

The idea behind this calculation, is to have resultant priorities issued from the existing ones. Instead of running through all the possibilities, we stay focused on the preferences given by the system at first place. This guarantees the respect of the first desired parameters.

Final Values of PEDASA Priorities. Finding the exact values of α_P, α_D and α_E should be based on a well-founded decision making. Accordingly, the main intent of this approach is to maximize the number of tasks respecting their deadlines without totally ignoring the delimited priorities. In fact, the predefined priority set does not allow the tasks to fully respect their deadlines. Hence, we aim for less than n tasks respecting the desired arrangement. Thus, while searching for the exact values of α_P, α_D and α_E, we start with a heuristic algorithm that runs through all the possibilities. Let $Vect_D$ and $Vect_P$ be the vectors containing, respectively, the number of tasks complying with their deadlines and initially fixed priorities under different values of the importance factors. Simply, these vectors refer to the number of elements of the Eqs. 10 and 12. Two constraints are considered while searching for the exact values of α_P, α_D and α_E: **(i)** when fixing these values we should have a number of tasks respecting their deadlines that is greater or equal to the number of tasks whose the old priorities correspond to the new ones, and **(ii)** the multiplication of these two numbers should be the maximum amongst all the possible values. The fact of considering the multiplication is based on the approximation to the logical operation *AND* exposed in [14]. This is mathematically described by Eq. 17. The choice behind this equation is founded on the desire of guaranteeing, so far, a higher number of tasks respecting their deadlines while partially obeying the predefined priorities. However, the complication occurs once the number of deadline-conducted tasks after calculation is lower than the one initially obtained by the FPP. This way, we focus on the set of priorities offered by this scheduling algorithm. Nevertheless, As long as there are tasks exceeding their deadlines, we proceed to changing the periods of the latter ones in order to have a fully functional system. The fundamental intention here consists on enhancing the values of a minimum number of periods related to the least prioritized tasks which outstrip their deadlines and replacing them by their multiples. Yet, the incrementation should obey to the constraint that we should not outpace the existing hyper-period.

$$\begin{aligned}\{Value(\alpha_P), Value(\alpha_D), Value(\alpha_E)\} = \{\alpha_P, \alpha_D, \alpha_E / \\ Vect_D(\alpha_P, \alpha_D, \alpha_E) \geq Vect_P(\alpha_P, \alpha_D, \alpha_E) \\ \wedge Vect_P(\alpha_P, \alpha_D, \alpha_E) * Vect_D(\alpha_P, \alpha_D, \alpha_E) = \\ max(Vect_D * Vect_P)\}\end{aligned} \tag{17}$$

5.2 Energy Consumption Adaptability

After finding the correct order in which the tasks should be executed based on the PEDASA computation, we introduce the concept of the system reloading. In fact, instead of manipulating the parameters of the set of tasks, it seems more efficient to reload the energy within the system, at a specific moment, in a way that its energy level gets restored. That is why in this part we search for the instants of energy reloading that enhance the lifespan of the whole system. Therefore, let t_{load} be the vector of instants, within a hyper-period T_{hyp}, at which the energy level attains 100 % again. This is analytically represented by Eq. 18.

$$\forall l, 0 < l \leq Size(t_{load}) : E(t_{load}(l)) = 100\,\% \tag{18}$$

Obviously, between the moments t and t_{load} the energy level can either decrease or remain the same. This depends on the energy required by each task. The chance of rewinding the system, allows the latter to execute all of its tasks without worrying about the exhaustion of the battery. We proceed in a determinist way, since the proposed scheduling algorithm is static. The first step consists on determining the instants at which each task got preempted by another one of a higher priority and the ones at which it resumed its execution. It is to mention that, in the real case, a task can be preempted one or several times during T_{hyp}. But it is also possible that the task never gets preempted. Generally speaking, let Pre_{ij} and Res_{ij} respectively be the instants of preemption and resumption number j for the task τ_i (i = 1..n). The second step takes in charge the definition of the new consumption functions that result from the several preemption cases that a task might have. For this matter, let $E_i^{''}(t)$ be the real consumption function that, based on $E_i^{'}(t)$, takes into consideration the preemption as well as the shift in the execution. This function is given by Eq. 19.

$$\forall i,j \in k_i : E_i^{''}(t) = \begin{cases} E_i^{'}(t + (S_i - A_i)) \; if \; t \geq S_i \vee t \leq Pre_{i1} \\ 0 \; if \; t \in \,]Pre_{ij}, Res_{ij}[\\ E_i^{'}(t + (S_i - A_i) - \sum\limits_{j=1}^{k_i} (Res_{ij} - Pre_{ij})) \; otherwise \end{cases} \tag{19}$$

The resulting function $E_{Ts}(t)$ (given by Eq. 20) represents the general survey on the energy consumption at any moment during T_{hyp}. This function is discontinuous and composed of the consumption functions related to each task. Let $\pi_{sys}(t)$ be the function that allows having an idea on the specific task executing at the instant t. This function is given by Eq. 21.

$$E_{Ts}(t) = \begin{cases} E_i^{''}(t) \; if \; \pi_{sys}(t) = \tau_i \\ 0 \; if \; \pi_{sys}(t) = 0 \end{cases} \tag{20}$$

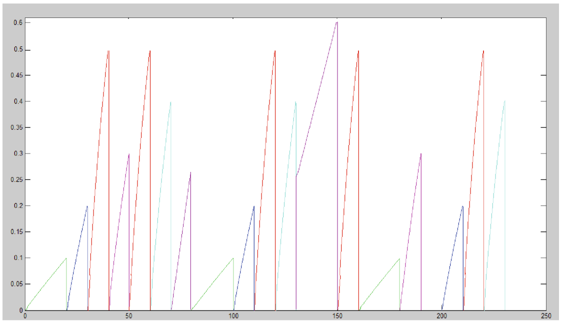
$$\pi_{sys}(t) = \begin{cases} \tau_i \; if \; \exists \tau_i / \pi_i(t) = 1 \\ 0 \; otherwise \end{cases} \tag{21}$$

Ultimately, the decrease in the available energy levels in the system during the same period, can be described by $E_{Sys}(t)$. Where this function (described by Eq. 22) is the difference between the available energy level at a precedent moment and the required energy at the moment that follows.

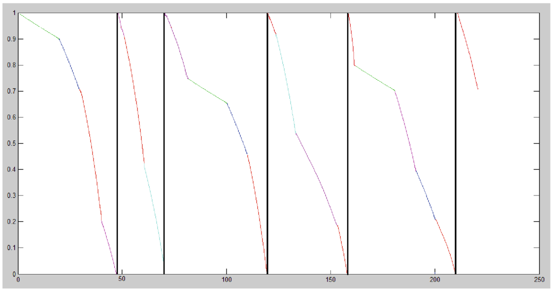
$$E_{Sys}(t) = E_{Sys}(t-1) - E_{Ts}(t) \tag{22}$$

Supposing that the procedure of reloading the battery is immediate and that the time it takes is insignificant, the reloading moment should more likely occur when $E_{Sys}(t)$ is zero. As a consequence, the system can dispose of a set of parameters ready for application without worrying about any future behavior since the periodicity is always predictable.

Running example: We consider the interval of time from 0 to 240 (the biggest period among all the tasks). During which, The task τ_1 got preempted only once. As a consequence, the following Figure exposes the function E_{Ts}, composed by the real consumption functions related to the tasks executing at that same time being.



The graphical representation of the available energy level, during the chosen time, conducted from the general energy consumption is given by the following Figure.



In fact, the graph demonstrates that the battery in this system needs to be reloaded around five times. Therefore the vector t_{load} is as follows (for the initially specified time in this running example).

$$t_{load} = \{48, 70, 119, 158, 210\}$$

6 Discussion

PEDASA is an optimal scheduling algorithm responsible of defining a new set of priorities without conducting an exhaustive calculation. In order to reduce this computation, we proceed to the strategy of limiting the possible priorities to the relation between the importance and the order factors. The aim of this strategy is to relate the new set of priorities to the first given parameters of the three criteria as well. The output of this algorithm is not only the new priorities, but also a new definition to the periods of the least prioritized tasks in a way that guarantees the feasibility of the scheduled task set. Moreover, the instants of reloading relevant to the battery are also resulted from the PEDASA algorithm. Compared to the mono-criteria scheduling algorithms (FPP, RM and LEF), the number of tasks respecting their deadlines and executing before battery exhaustion (Eqs. 9 and 11) is way important. However, even if the range of the tasks under the new set priorities is different from the initial ones it is still substantial compared to the RM and LEF algorithms. Through Fig. 1, where we consider the case study displayed in Sect. 3, it is noticeable that the surface of the PEDASA consideration is larger than that of RM, FPP or LEF schedulers which pinpoints its remarkable contribution. Additionally, compared to the existing multi-criteria real-time scheduling algorithms that are based on fuzzy logic, PEDASA offers a more determinist method to find the new adequate set of priorities and adjusts several parameters to enhance the performance of the system. Our approach is also reversible. This means that it can simply refer to a RM, FPP or LEF scheduling if the result that one of them offers is effectively performed. Consequently, it economizes in terms of energy and time computation.

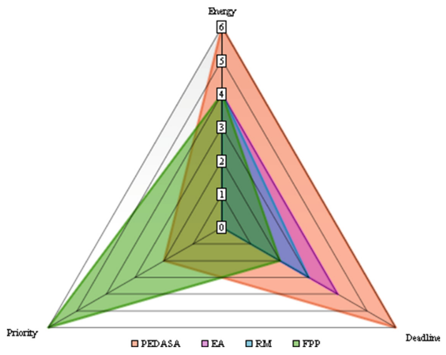


Fig. 1. Respect ratio of the parameters energy, deadline and initial priorities.

7 Conclusion

In this paper we introduced PEDASA as a static soft real-time scheduling algorithm that solves the multi-criteria decision making in a computational way. The performance of this algorithm is then compared with that of FPP, RM and LEF. It is put in display that our proposed approach not only confirms an important performance compared to these algorithms but also changes the system into a more versatile structure. Surely, the establishment of power reloading possibility in predetermined instants guarantees the continuous functioning of the system. Similarly, the feasibility of the task set is very accurate with the purpose of the real-time constraints. Therefore, this algorithm could be more appropriate for use on the real-time systems that are monitored by users and that require a consistent power level. As a perspective, we wish to apply this new algorithm to a system where the three criteria priority, deadline and energy efficiency are very important features and can be affected by different reconfiguration scenarios. The wireless sensor network is a potential application of this work [3,4].

References

1. Baccouche, L., Eleuch, H.: Rt-Dbp: a multi-criteria priority assignment scheme for real-time tasks scheduling. Appl. Math. **6**(2), 383–388 (2012)
2. Fahmy, M.: A fuzzy algorithm for scheduling non-periodic jobs on soft real-time single processor system. Ain Shams Eng. J. **1**(1), 31–38 (2010)
3. Gasmi, M., Mosbahi, O., Khalgui, M., Gomes, L.: New pipelined-based solutions for optimal reconfigurations of real-time systems. In: Proceedings of the European Simulation and Modelling Conferences (2014)
4. Gasmi, M., Mosbahi, O., Khalgui, M., Gomes, L.: Reconfigurable priority ceiling protocol under rate monotonic based real-time scheduling. In: 2014 11th International Conference on Informatics in Control, Automation and Robotics (ICINCO), vol. 1, pp. 42–52. IEEE (2014)
5. Kopetz, H.: Real-Time Systems: Design Principles for Distributed Embedded Applications. Springer, New York (2011)
6. Lehoczky, J., Sha, L., Ding, Y.: The rate monotonic scheduling algorithm: exact characterization and average case behavior. In: 1989 Proceedings of Real Time Systems Symposium, pp. 166–171. IEEE (1989)
7. Lindberg, P., Leingang, J., Lysaker, D., Bilal, K., Khan, S.U., Bouvry, P., Ghani, N., Min-Allah, N., Li, J.: Comparison and analysis of greedy energy-efficient scheduling algorithms for computational grids. In: Zomaya, A., Lee, Y.C. (eds.) Energy Aware Distributed Computing Systems. Wiley, Hoboken (2012)
8. Park, S., Kim, J.H., Fox, G.: Effective real-time scheduling algorithm for cyber physical systems society. Future Gener. Comput. Syst. **32**, 253–259 (2014)
9. Peng, B., Fisher, N., Bertogna, M.: Explicit preemption placement for real-time conditional code via graph grammars and dynamic programming. Ph.D. thesis, Wayne State University (2014)
10. Rattanatamrong, P., Fortes, J.A.: Fuzzy scheduling of real-time ensemble systems. In: 2014 International Conference on High Performance Computing & Simulation (HPCS), pp. 146–153. IEEE (2014)
11. Salmani, V., Ensafi, R., Khatib-Astaneh, N., Naghibzadeh, M.: A fuzzy-based multi-criteria scheduler for uniform multiprocessor real-time systems. In: 10th International Conference on Information Technology, (ICIT 2007), pp. 179–184. IEEE (2007)
12. Shin, K.G., Ramanathan, P.: Real-time computing: a new discipline of computer science and engineering. Proc. IEEE **82**(1), 6–24 (1994)
13. Wang, Y., Saksena, M.: Scheduling fixed-priority tasks with preemption threshold. In: 1999 Sixth International Conference on Real-Time Computing Systems and Applications, RTCSA 1999, pp. 328–335. IEEE (1999)
14. Zadeh, L.A.: The Concept of a Linguistic Variable and Its Application to Approximate Reasoning. Springer, New York (1974)