mathematics

MDPI

Article

A Hardware-Aware Application Execution Model in Mixed-Criticality Internet of Things

Cristina Sorina Stângaciu [1,*], Eugenia Ana Capota [1], Valentin Stângaciu [1], Mihai Victor Micea [1] and Daniel Ioan Curiac [2,*]

[1] Computer and Information Technology Department, Politehnica University, Vasile Parvan 2, 300223 Timisoara, Romania; eugenia.capota@cs.upt.ro (E.A.C.); valentin.stangaciu@cs.upt.ro (V.S.); mihai.micea@cs.upt.ro (M.V.M.)

[2] Automation and Applied Information Department, Politehnica University, Vasile Parvan 2, 300223 Timisoara, Romania

* Correspondence: cristina.stangaciu@cs.upt.ro (C.S.S.); daniel.curiac@aut.upt.ro (D.I.C.)

Abstract: The Real-Time Internet of Things is an emerging technology intended to enable real-time information communication and processing over a global network of devices at the edge level. Given the lessons learned from general real-time systems, where the mixed-criticality scheduling concept has proven to be an effective approach for complex applications, this paper formalizes the paradigm of the Mixed-Criticality Internet of Things. In this context, the evolution of real-time scheduling models is presented, reviewing all the key points in their development, together with some connections between different models. Starting from the classical mixed-criticality model, a mathematical formalization of the Mixed-Criticality Internet of Things concept, together with a specifically tailored methodology for scheduling mixed-criticality applications on IoT nodes at the edge level, is presented. Therefore, a novel real-time hardware-aware task model for distributed mixed-criticality systems is proposed. This study also offers a model for setting task parameters based on an IoT node-related affinity score, evaluates the proposed mapping algorithm for task scheduling, and presents some use cases.

Keywords: distributed computing; scheduling; scheduling algorithm

MSC: 68M20; 68M14

Citation: Stângaciu, C.S.; Capota, E.A.; Stângaciu, V.; Micea, M.V.; Curiac, D.I. A Hardware-Aware Application Execution Model in Mixed-Criticality Internet of Things. *Mathematics* **2022**, *10*, 1537. https://doi.org/10.3390/math10091537

Academic Editor: Marina Alexandra Pedro Andrade

Received: 1 April 2022
Accepted: 29 April 2022
Published: 3 May 2022

1. Introduction

In the era of telecommunication and interconnectivity, the Internet of Things (IoT) is a promising new technology for connecting intelligent objects that surround us into a large network, often distributed over large geographic regions [1,2]. Real-time and non-real-time, critical, and non-critical systems coexist with increasingly complex and sometimes conflicting demands from the network. Thus, new concepts for the provisioning, management, and monitoring of these systems and their components must be developed. The development of such concepts has led to an entirely new field, the Real-Time Internet of Things (RT-IoT) [3], which promises better connectivity and efficient use of next-generation embedded devices. As the field of real-time systems has developed in recent years, it is natural to integrate new concepts from RT systems into RT-IoT. This concept, which is currently gaining attention in classical RT systems, is the concept of Mixed-Criticality Systems (MCSs), which define systems running real-time tasks of different criticality levels sharing the same platform. As many critical real-time applications have already been implemented using distributed heterogeneous architectures [4], the MCS concept can also be applied to RT-IoT, leading to a new scheduling paradigm called MC-IoT (for a full list of abbreviations, please see Appendix B).

The need for comprehensive solutions that integrate systems with strict timing constraints/requirements into IoT architectures is not new and has been extensively discussed in [5]. This need is also reflected in the design and implementation of real-time/time-triggered Ethernet protocols with mixed-criticality support [4]. However, only a few examples of real-time systems in their classical form or mixed-criticality systems have been integrated into IoT architectures. In [1], the placement of services and data in fog devices is discussed with regard to the optimization of the system. The challenges of both soft and hard real-time systems are addressed in [3] by introducing the term RT-IoT. Papers such as [6] and [7] present examples of autonomous IoT devices and applications, while [8] presents an example of a collaborative homogeneous IoT system, and [9], a system development platform for reducing the complexity of developing IoT-enabled applications with different criticalities. In [10], the deep learning approach is used to schedule real-time edge services, while Jin et al. [11] proposed a mechanism for data aggregation in mobile privacy-aware crowdsourcing systems. Furthermore, a real-time system that proposes the approximate image sharing and energy-aware adaptation to obtain higher bandwidth and energy efficiency, called BEES, was introduced in [12].

From our perspective, MC-IoT can effectively address the problem of resource management at the edge level of IoT platforms running a variety of applications with different criticalities and different time requirements, such as sensor fusion for automated driving applications [13], smart buildings [14], and healthcare mixed-criticality applications [15]. The scheduling problem in real-time distributed heterogeneous systems is not trivial and has been insufficiently addressed, despite its potential [16]. A few papers addressing this problem have been published [17–20].

The heterogeneous nature of the IoT architecture has a direct influence on applications that run on these different hardware components.

- The application code execution time is strongly influenced by the target system running the application.
- In the same manner, the power consumption owing to the execution of the same code may be different for target j than for target k.

Therefore, new application models are required for the MC-IoT applications. These models need to be considered in addition to the temporal and sometimes criticality aspects as well as hardware particularities.

In RT systems in general and in RT-IoT in particular, temporal behavior is a significant aspect, as important as the correctness and accuracy of the provided result [21]. Thus, the application models for MC-IoT, as a particular case of RT-IoT, should consider the influence of the hardware particularities.

In this study, we formalized the basic concepts of MC scheduling in the IoT domain and defined an MC–IoT paradigm. We also provide a methodology for scheduling MC applications in MC-IoT.

The main contributions of this research are:

- Definition and mathematical formalization of the Mixed-Criticality Internet of Things scheduling concept;
- The scheduling problem in MC-IoT and RT-distributed systems was mathematically formulated;
- A novel hardware-aware real-time task model for distributed mixed-criticality systems such as MC-IoT was proposed;
- A methodology for mapping tasks on heterogeneous targets, considering both timing and hardware particularities, was proposed and analyzed.

The evolution of real-time scheduling models is presented in Section 2, reviewing all the key points in their development. We created a mathematical formalization of the scheduling problem in Section 3. We adapted the classical MC task model to the MC-IoT paradigm in Section 4. The methodology for scheduling tasks in MC-IoT is proposed together with example algorithms for setting task parameters and distributing tasks on

different IoT nodes in Section 5. The proposed function for task distribution is evaluated and compared with the classical mapping function in Section 6. The article ends with conclusions and future perspectives in Section 7.

2. Related Work

Tasks are basic execution units of an application. Depending on their activation patterns, they can be periodic or sporadic. Periodic tasks are activated at a constant rate [22], whereas sporadic tasks have a minimum inter-arrival time [23,24]. In the mixed-criticality paradigm, each task has an assigned criticality level and a set of properties [25,26] (for a full list of symbols, please see Appendix B).

$$\boldsymbol{\tau_i} = \left\{ \boldsymbol{T_i}, \boldsymbol{D_i}, \boldsymbol{L_i}, \left\{ \boldsymbol{C_{i,j}} \middle| \boldsymbol{j} \in 1 \ldots \boldsymbol{l} \right\} \right\} \tag{1}$$

where l represents the number of criticality levels, T_i—the period for periodic tasks or the (minimum) arrival interval between two consecutive jobs of the same task i, D_i is the time by which any job execution needs to be completed relative to its release time, L_i is the criticality level (1 being the lowest level), and $C_{i,j}$ is the computation time (vector of values—one per criticality level j, for levels lower or equal to the criticality level L_i ($j \leq L_i$) expressing the worst-case execution time for each criticality level).

A task consists of a series of jobs, with each job inheriting the set of parameters of the task (T_i, D_i, L_i) to which it adds its parameters [27]. Thus, the k-th job of task τ_i is characterized as:

$$\boldsymbol{J_{i,k}} = \left\{ \boldsymbol{a_{i,k}}, \boldsymbol{d_{i,k}}, \boldsymbol{c_{i,k}}, \boldsymbol{T_i}, \boldsymbol{D_i}, \boldsymbol{L_i} \right\}) \tag{2}$$

where $a_{i,k}$ represent the arrival time $a_{i,k+1} - a_{i,k} >= T_i$; $d_{i,k}$ represents the absolute deadline ($d_{i,k+1} = a_{i,k} + D_i$); $c_{i,k}$ represents the execution time, which is dependent on the criticality mode of the system (for L_j, $c_{i,k} = C_{i,j}$); and T_i, D_i, L_j have the same meaning as in the task model.

The classical model has been the source of other simplified models, such as those introduced by Burns [28]. In [29], it was proven that Vestal's model is a generalization of Burns' model. Another classical simplification of the model considers only two levels of criticality (LO and HI) [30].

In time, other parameters were transformed from a simple scalar value to a vector of values depending on the criticality level. Such is the case of the period in [31]. In the same manner, the worst-case memory access was considered and added to the general model as a vector of values in [32], and a vector of values describing the QoS was added in [33]. In [34], a new model called Elastic Mixed-Criticality (E-MC) was introduced, which introduces the idea of variable periods for low-criticality tasks.

In addition to general mixed-criticality models derived from hard real-time task models, another task model for cyber-physical mixed-criticality systems was proposed in [35]. This model can be considered as a translation of the model presented in [36] for real-time tasks into an MCS task model by expressing the number of tolerable deadline misses as a function of the criticality level instead of a constant value, as in the classical real-time case.

On the one hand, we have the task-level models presented in the previous paragraph, which are mostly used for sets of independent tasks and are insufficient for sets of tasks with precedence constraints. However, we have models derived from graph-based real-time task models [37], such as those proposed in [38]. The mode-switching Diagraph Real-Time (MS-DRT) task model is such an example, which was proposed by Ekberg and Yi in [39]. These types of models are more complex and describe, in addition to temporal behavior, functional dependencies between tasks. Another graph-based model that specifies the allowed interference between tasks, called interference constraint graph (ICG), was proposed in [40]. The task parameters are the same as those in the Vestal model, and the main difference lies in the fact that the graph models the relationship between tasks, namely, task interference.

However, we have graph-based models for which we could not identify a central influential model but encountered several variants, from which we mention Ekberg and Yi (2016) and Huang et al. (2013).

Figure 1 summarizes the above paragraphs, presenting the main task models for RTS (real-time systems) and MCS (mixed-criticality systems), their main differences, and their evolution. For uniformity reasons, the parameter representing the criticality level has been marked with L even if in some of the original articles other symbols were used. Vestal's model (2007) is the most influential model, which was developed from the sporadic task model proposed by Mok in 1983 and was further customized into several simplified models, among which we consider worth mentioning the Burns and Baruah model (2013). In addition to general task models, there are models specifically for firm/soft real-time tasks with criticality levels (Lee and Shin, 2017).

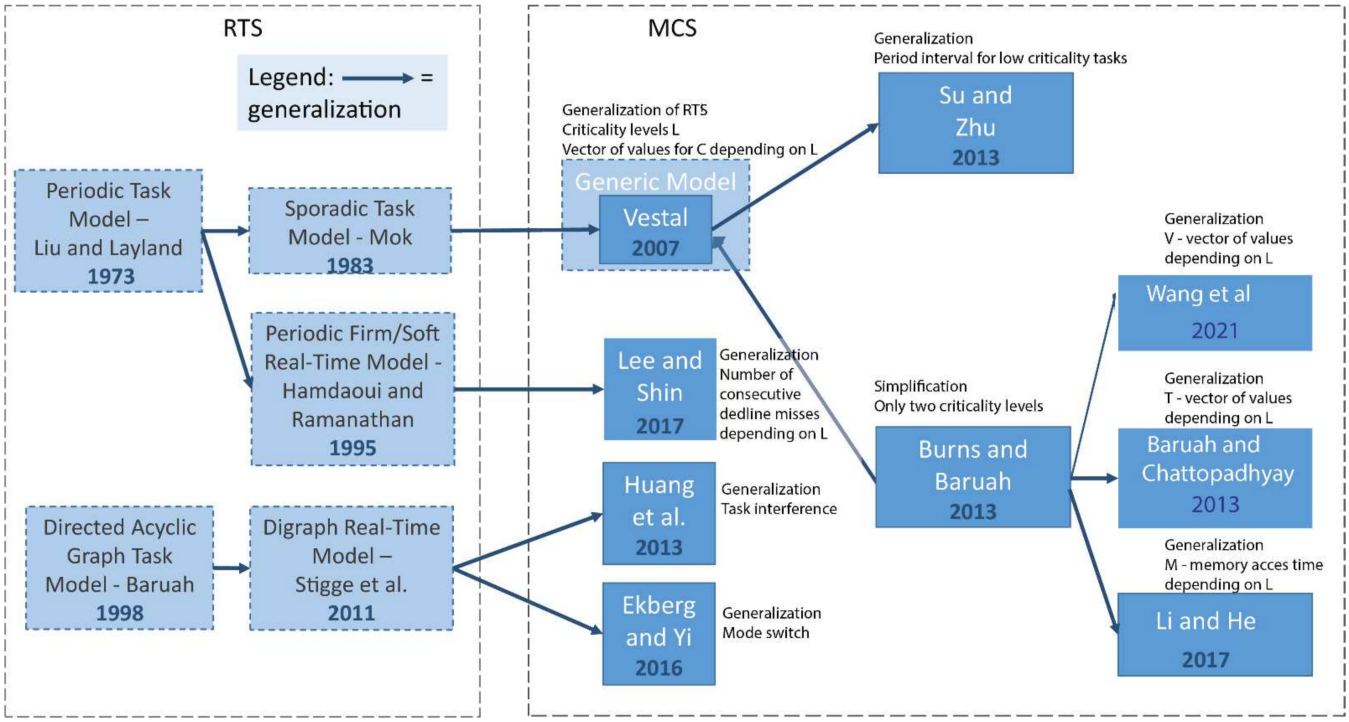

Figure 1. Task Models Evolution.

Although there are a relatively large number of task models, there are only a few core models that give birth to different variations. Most of the variations are compatible with one another, but they focus on different aspects, generalizing or simplifying certain behaviors/parameters, thus differing in the assumptions regarding the task parameters (see Appendix A).

3. Application Execution Model and Scheduling Problem Formulation

With the development of 6G networks, the IoT applications running at the edge level are integrating more and more real-time and mixed-criticality functionalities [33].

Mixed-Criticality Systems represent a special class of real-time systems in which application functions with different criticalities share the same resources in terms of computation and/or communication [12]. The MCS concept offers important advantages when integrated into different types of systems that directly interact with the environment [12]. IoT is also a type of system that interacts directly with the user and/or environment. As IoT systems are rapidly evolving towards real-time IoT systems, where tasks with different time constraints and criticality levels coexist, it is only natural that the application scheduling

in such systems becomes increasingly complex, and the need to integrate the concept of mixed-criticality becomes stringent.

Adapting the MCS concept to IoT architectures gives birth to a new scheduling paradigm called MC-IoT, for which we propose the following definition.

Definition 1. *Mixed Criticality-Internet of Things (MC-IoT) is a type of system that run real-time tasks of different criticalities at the edge level of IoT architectures.*

In MCSs, as in any real-time system, applications are split into basic units of execution called tasks. The scheduling issue of these tasks on multiprocessors and distributed systems have become prominent in RT-IoT systems such as 6G networks [33].

Tasks can be independent of one another or have a precedence dependence on different tasks. Each task or set of dependent tasks is allocated to a specific node in a process called global level scheduling, and each set of independent or dependent tasks is scheduled at the device level in a process called local scheduling.

Thus, the scheduling problem at the edge level can be divided into two sub-problems:

(1) Task mapping (at an intermediate/Fog level): Each task must be allocated or mapped to a processing element.
(2) Local task scheduling (at the device/Edge node level): all tasks allocated to a certain processing element must be schedulable.

Until now, the task mapping problem was not approached by taking into consideration the hardware heterogeneity. Thus, we further propose a hardware-aware solution to this problem.

In this paper, we assume the following hypotheses:

The system is represented by an IoT layer architecture with heterogeneous edge nodes. The architecture contains three main computation levels (Edge, Fog, Cloud) [41].

These applications were implemented as systems for periodic/sporadic independent mixed-criticality tasks.

These tasks have stringent timing requirements (i.e., hard real-time tasks).

The worst-case execution time (WCET) was determined via static analysis for high criticality levels and estimated through measurements for low criticality levels.

Each task was statically assigned to a processing element (PE) and could not migrate during runtime.

Each processing element starts in the lowest criticality mode, L_1, and remains in that mode as long as all jobs are executed within their criticality level computation times $C_{i,1}$. If any job is executed for more than $C_{i,1}$ without signaling completion, the system immediately changes to L_2 and so on [25].

For simplicity, we consider that, for each criticality mode L_j, tasks with a criticality lower than L_j are dropped [25]. Still, the models and partitioned algorithms proposed in this article are valid, even if we consider more complicated local task scheduling algorithms, which do not drop lower criticality tasks but schedule them using processor slack time.

The system is heterogeneous, which means that the task execution time varies depending on the PEs.

4. The Proposed MC-IoT Task Model

As was presented in Section 2, the current MC task models do not take into consideration the hardware heterogeneity of the system.

We begin with the classical MCS model proposed by Vestal in 2007 [25] and consider the parameters from Equation (1). In addition to the temporal behavior already included in this model, we propose an extension consisting of a new parameter, affinity score A_i, which is defined as a vector of values, one per host, representing the affinity score of the task for each host. The affinity score is an integer between zero and p, where p is the number of processing elements. A higher value indicates higher affinity, and zero indicates no affinity.

Moreover, we propose to express the computation time not only as a function of the criticality level (L_i) but also as a function of the processing hardware element running the task (PE_q). Thus, task model (1) becomes:

$$\boldsymbol{\tau_i} = \left\{ \boldsymbol{T_i}, \boldsymbol{D_i}, \boldsymbol{L_i}, \left\{ \boldsymbol{C_{i,j,q}} \middle| \boldsymbol{j} \in 1 \ldots \boldsymbol{l}, \boldsymbol{q} \in 1 \ldots \boldsymbol{p} \right\}, \left\{ \boldsymbol{A_{i,q}} \middle| \boldsymbol{q} \in 1 \ldots \boldsymbol{p} \right\} \right\} \tag{3}$$

where C_i is a matrix of size $p \times l$ (p represents the number of PEs, and l is the number of criticality levels).

The task affinity score can be set statically by the task creator or computed using an algorithm based on the resources required by the task and the task computation time for different processing elements. If a task cannot be scheduled on a particular PE, the WCET value can be set to infinity, which translates to an affinity score of zero.

In the following analysis, we consider a dual criticality system. However, these algorithms can also be extended to platforms with multiple criticality levels. In criticality-aware heuristics, it makes more sense to use utilization in the high criticality (Hi) mode on a certain processing element q during the assignment of Hi-criticality tasks and utilization in the low criticality (Lo) mode during the assignment of Lo-criticality tasks. Following this principle, the affinity score is set for Hi-criticality tasks using their Hi-criticality WCET and for Lo-criticality tasks using their Lo-criticality WCET.

5. A Methodology for Mapping Tasks on Different Processing Elements

Having the previously defined MC-IoT task model and scheduling problem formalized in Section 4, we propose a hardware-aware methodology for mapping tasks into our proposed MC-IoT architecture. The methodology comprises different methods for setting the newly introduced task parameter, namely the affinity score, and defining a suitable mapping function to respect both application and resource constraints.

5.1. Setting Affinity Score

While the task parameters inherited from the classical MCS model are assigned considering only the temporal behavior [25], the affinity score is computed by considering the particularities of the processing elements. This can be achieved using the following methodology.

5.1.1. Set Affinity Score Based on Computation Time

The affinity score can be set based on the computation time using the algorithm presented in Algorithm 1.

Algorithm 1: SetAffinity_WCET

Input: $C_{i,\,2,q}$
Output: $A_{i,\,q}$

```
for i ∈ {1, 2, ..., n} do
    for q ∈ {1, 2, ..., p} do
        X_{i,2,q} ← C_{i,2,q}
    end for
end for
for i ∈ {1, 2, ..., n} do
    a ← 1
    for q ∈ {1, 2, ..., p} do
        index ← max(X_{i,2,q})
        A_{i,index} ← a
        a ← a + 1
        X_{i,2,index} ← 0
    end for
end for
```

Step 1: *Extract the task computation time for the highest criticality level (2 in the case of a dual criticality system) on each PE (the second column in the generated computation time matrix $C_{i,j,q}$) by copying it into an array of structures $X_{i,2,q}$. Each structure has a computation time value $X_{i,2,q}$, and a PE index (q), where i represents the task number, which is fixed, and 2 represents the highest criticality level. Index q varies from one to p.*

Step 2: *Extract the matrix line index of the maximum value for $X_{i,2,q}$ from the array when q varies from one to p.*

Step 3: *In the affinity array, set the affinity score $A_{i,q}$ to 1 for the PE corresponding to the highest $X_{i,2,q}$ value, and then 2 for the PE corresponding to the highest $X_{i,2,q}$ from the array after setting the element with the highest value from the first iteration $X_{i,2,q}$ to 0, 3 for the remaining highest value, and so on, while $q \leq p$.*

5.1.2. Set Affinity Score Based on Criticality Level

To distribute tasks on different processing elements considering the criticality level of the task, we can employ carefully selected algorithms, such as those described in [42].

An alternative method for mapping the tasks that consider the affinity score is proposed as Algorithm 2.

Algorithm 2: SetAffinity_criticality ($p > l$)

Input: L_i, $C_{i,\,2,q}$
Output: $A_{i,\,q}$

for $i \in \{1,\,2,\,\ldots,\,n\}$ **do**
  for $q \in \{1,\,2,\,\ldots,\,p\}$ **do**
    $X_{i,2,q} \leftarrow C_{i,\,2,q}$
  end for
end for
for $q \in \{1,\,2,\,\ldots,\,p\}$ **do**
  $PE_{q,\,j} \leftarrow mod(q,\,l)$
  if $mod(q,\,l) = 0$ **then**
    $PE_{q,\,j} \leftarrow l$
  end if
end for
for $i \in \{1,\,2,\,\ldots,\,n\}$ **do**
  $a \leftarrow 0$
  for $q \in \{1,\,2,\,\ldots,\,p\}$ **do**
    $\{maxVal, index\} \leftarrow \max(X_{i,\,2,q} \,|\, PE_{q,\,j} \sim= L_i)$
    if $maxVal \sim= 0$ **then**
      $a \leftarrow a + 1$
      $A_{i,\,index} \leftarrow a$
      $X_{i,2,index} \leftarrow 0$
    end if
  end for
end for
for $i \in \{1,\,2,\,\ldots,\,n\}$ **do**
  $a2 \leftarrow p - a$
  for$q \in \{1,\,2,\,\ldots,\,p\}$ **do**
    $\{maxVal, index\} \leftarrow \max(X_{i,\,2,q} \,|\, PE_{q,j} = L_i)$
    if $maxVal \sim= 0$ **then**
      $a2 \leftarrow a2 + 1$
      $A_{i,\,index} \leftarrow a2$
      $X_{i,\,2,index} \leftarrow 0$
    end if
  end for
end for

Step 1: *Build an array of structures $PE_{q,j}$. Each structure has a criticality level L_j that represents the expected criticality level of the tasks to be partitioned on PE_q and PE index (q). Index q varies from 1 to p, whereas L_j is given by computing $L_j = PE_q$ mod l, where l is the number of criticality levels and mod represents the modulo operation. If L_j is zero, we consider $L_j = l$.*

Step 2: *Assign each task τ_i according to criticality level L_i. We have two subsets: PEs with expected criticality $PE_{q,j}$ equal to $L_{i,}$ and PEs with expected criticality $PE_{q,j}$ not equal to L_i. The affinity scores $A_{i,q}$ had the highest values for the first subset of PEs. For each subset of PEs, an affinity score was assigned according to the computation time, as shown in Algorithm 1.*

If the number of criticality levels exceeded the number of PEs, the affinity score was set according to Algorithm 3.

Algorithm 3: SetAffinity_criticality (l>p)

Input: L_i, $C_{i,2,q}$
Output: $A_{i,\ q}$

```
for i ∈ {1, 2, ..., n} do
    for q ∈ {1, 2, ..., p} do
        X_{i, 2,q} ← C_{i,2,q}
    end for
end for
for i ∈ {1, 2, ..., n} do
    q ← mod(L_i, p)
    if q = 0 then
        q ← p
    end if
    A_{i,q} ← p
    X_{i,l,q} ← 0
end for
for i ∈ {1, 2, ..., n} do
    a ← 0
    for q ∈ {1, 2, ..., p − 1} do
        index ← max(X_{i, l,q})
        a ← a + 1
        A_{i, index} ← a
        X_{i, l,index} ← 0
    end for
end for
```

Step 1: *For each task τ_i, obtain the PE on which it is expected to run by computing $PE_q = L_j$ mod p, where p is the number of PEs, and mod represents the modulo operation. If PE_q is 0, we consider $PE_q = p$. Next, we set the affinity score $A_{i,q}$ for PE_q as p.*

Step 2: *For each task τ_i, set the remaining affinity scores $A_{i,q}$ according to the computation time (Algorithm 1), where q ranges from 1 to p-1.*

5.2. Task Mapping

As highlighted in the literature [33], the partitioned scheduling approach can achieve better schedulability results than the global scheduling approach. Thus, we also propose a partitioned approach.

Given a set of independent MC tasks and processing elements, the problem involves determining the following function:

$$M : \tau \rightarrow P \tag{4}$$

where τ is the set of tasks in the system, P is the set of processing elements, and M is a morphism.

Considering the working hypotheses presented above, tasks cannot migrate during runtime; therefore, each task was assigned to a single PE. $M(\tau_i)$ represents the PE at which τ_i runs. Figure 2 illustrates the assignment of tasks to PEs, where p is the total number of PEs, and n is the number of tasks in the system.

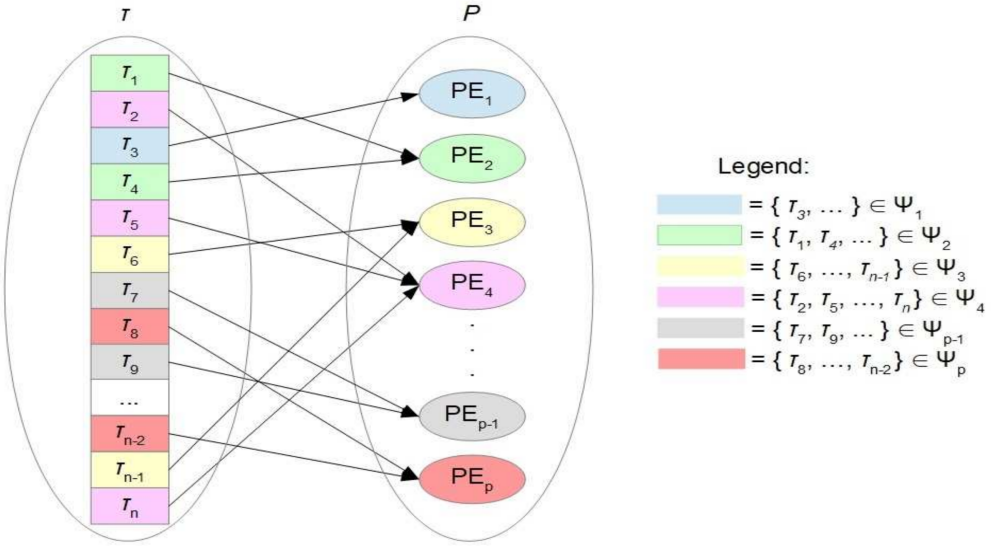

Figure 2. Assignment of Tasks to Processing Elements.

After each task is assigned to a processing element, the tasks allocated to PE_q form a task subset defined as Ψ_q [27]. If task set τ is successfully partitioned, then

$$\boldsymbol{\tau} = \boldsymbol{\Psi}_1 \cup \boldsymbol{\Psi}_2 \cup \boldsymbol{\Psi}_3 \cup \ldots \cup \boldsymbol{\Psi}_p \tag{5}$$

where Ψ_q can also be an empty subset {Ø}.

The goal is to find a suitable M function to comply with the next request given the following priority order:

(1) Suitable subsets of tasks Ψ_q are created, such that each subset is schedulable by the local scheduling algorithm running on its assigned PE_q.
(2) Respect task affinity score.
(3) Optimize resource usage.

Owing to its increased complexity, the mapping function should be implemented at the intermediate (Fog) level.

5.2.1. Local Task Scheduling

According to Vestal [25], a task set is schedulable if the deadline of each task τ_i is greater than or equal to its worst-case response time.

$$\boldsymbol{R_i} \leq \boldsymbol{D_i} \tag{6}$$

where WCET is the maximum duration between completion and activation of each job of task τ_i [43].

$$\boldsymbol{R_i} = \boldsymbol{C_i} + \boldsymbol{I_i} \tag{7}$$

where I_i is the inference from higher priority tasks.

The value of R_i is dependent on the scheduling algorithm and can be iteratively computed as the least fixed point of (8), where *hp(i)* is a subset of tasks with a priority

higher than or equal to that of task τ_i but not containing τ_i, C_i is the worst-case execution time, and T_i is the task period [25]:

$$R_i = \sum_{\tau_j \epsilon hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{8}$$

Scheduling in mixed-critic systems is feasible if the following two conditions are satisfied [44].

Condition 1. If all jobs run no more than their $C_{i,j}$ values for the current L_j criticality mode, then all jobs with a criticality level L_i higher than or equal to L_j must be completed before their deadline.

Condition 2. If at least one job exceeds its $C_{i,j}$ execution time value for the current L_j criticality mode, then the current criticality mode changes to the next criticality value (L_{j+1}), for which Condition 1 must also be true.

Thus, Condition 1 must be satisfied for all the criticality modes available in the system. For a k criticality mode (L_k) on the PE_q processing element, Equation (8) becomes:

$$R_{i,k,q} = \sum_{\tau_j \epsilon hp(i)} \left\lceil \frac{R_{i.k.q}}{T_j} \right\rceil C_{j,k,q} \tag{9}$$

Checking Condition 1, for the L_k criticality mode, reduces to the verification of Condition (6), particularly for the L_k criticality mode on the PE_q processing element:

$$R_{i,k,q} \leq D_i \tag{10}$$

is true for all the tasks in the task set.

Moreover, for all deadlines to be satisfied, the processing units must not be overloaded with the tasks. Thus, a necessary but sufficient condition bound on the load metric for any processing unit is given by [44]:

For each $q = 1 \ldots$ p:

$$U_{L_j \Psi_q} \leq 1, \quad j = 1 \ldots l \tag{11}$$

For periodical/sporadic tasks, the total processor utilization in execution mode L_j is:

$$U_{L_j \Psi_q} = \sum_{\tau_i \epsilon hc(L_j),\ k=j}^{k \leq l} \frac{C_{i,k,q}}{T_i} \tag{12}$$

where $hc(L_j)$ is a subset of tasks from Ψ_q with criticality higher than or equal to L_j and l is the number of criticality levels. $L_{k=j \ldots l}$ represents the criticality levels higher than or equal to L_j, *and* $C_{i,k,q}$ represents the WCET of task i, which runs in criticality mode k on the processing element PE_q.

Moreover, many local task-scheduling algorithms have sufficient conditions. For example, a sufficient condition for a local task scheduling algorithm called EDF-VD when the system has two criticality levels has been proven to be [27].

$$max\left(U_{L_1 \Psi_q}, U_{L_2 \Psi_q}\right) \leq \frac{3}{4} \tag{13}$$

5.2.2. Affinity Score Deviation Function

The task affinity score can be set statically by the task creator or computed using an algorithm similar to that proposed in Section 5.1, based on the resources needed by the task and the task computation time for different processing elements. The mapping function should also consider the affinity score for each task, provided that the local feasibility

conditions are not violated. To evaluate this, we propose a function representing the total affinity-score deviation $A_{d\tau}$ for a particular mapping as follows:

$$\boldsymbol{A_{d\tau}} = \sum_{\boldsymbol{i}=1}^{\boldsymbol{n}} (\boldsymbol{p} - \boldsymbol{A_{i,q}}) \tag{14}$$

where q is the index of subset Ψ containing task i and p is the number of PEs (p is also equal to the highest affinity score value).

Our goal is to minimize A_{dt} with respect to the feasibility conditions imposed by the local scheduler (Condition 1).

5.2.3. Proposed Mapping Function—Best Affinity First

Based on the methods used to set the affinity score introduced in the first paragraphs of this section, we developed a mapping algorithm (see Algorithm 4) that considers the affinity value when partitioning tasks to PEs, namely Best Affinity Fit (BAF). This algorithm is evaluated later in the experimental section.

Algorithm 4: Best Affinity Fit (BAF)

Input: τ_i
Output: Ψ_q, $U_{L_j\Psi_q}$

```
for i ∈ {1, 2, ..., n} do
    assign ← 1
    {maxVal, index} ← max(A_{i,q})
    while U_{L_jΨ_index} + C_{i,j,index}/T_i > 1 do
        A_{i,index} ← 0
        {maxVal, index} ← max(A_{i,q})
        if maxVal = 0 then
            assign ← 0
        end if
    end while
    if assign ← 1 then
        add(Ψ_index, τ_i)
        U_{L_jΨ_index} ← U_{L_jΨ_index} + C_{i,j,q}/T_i
    end if
end for
```

For each task τ_i:

Step 1: *First, we assume that the task can be assigned to PE. Therefore, we set the variable assigned to 1. We find the PE with the highest affinity score for task i, which has enough space to accept task* τ_i *(PE utilization must respect Equation (14)).*

Step 2: *If a certain* PE_{index} *does not have sufficient space for task i, the affinity score* $A_{i,index}$ *to* 0*. If there are no PEs that can host task i, then the set is assigned to 0.*

Step 3: *If we have found a* PE_{index} *which can accept task* τ_i*, then add task i to subset* Ψ_{index} *and update the PE utilization.*

6. Examples and Comparative Evaluation

We conducted a series of simulation experiments to evaluate the effectiveness of our mapping technique: the Best Affinity Fit. It has been compared against two other relevant mapping methods, Best Fit Decreasing Utilization (BFDU) and Best Fit Decreasing Criticality (BFDC), which are some of the most frequently used algorithms [45]. BAF dispatches tasks to processors according to the Affinity value, while BFDU and BFDC tasks are first ordered by decreasing utilization for BFDU or by decreasing criticality for BFDC and then assigned to each processor. The processors are also ordered by their decreasing

utilization. Two affinity assignment strategies were also evaluated: one allocates values according to the WCET, whereas the other depends on the criticality level.

6.1. Random Task Set Generation

All tasks were randomly generated in Matlab (R2018b) using the task set generation algorithm introduced in [46], which is a slight modification of the workload generation algorithm proposed by Guan et al. [47]. We consider a dual-criticality system *{Lo, Hi}*. Each new task, τ_i, was generated as follows:

Task criticality level: $L_i = Hi$ with probability P_{Hi}; otherwise, $L_i = Lo$.

Task period: T_i is a randomly generated value drawn from a uniform distribution [10, 100].

Task deadline: $D_i = T_i$, because of the implicit deadline constraint.

The utilization of each task $U_{i,j,q}$ is a matrix $p \times l$, where p is the number of PEs in the system and l represents the number of criticality levels. To generate the utilization values, we considered five input parameters [46].

U_{bound}:

$$\max(U_{Lo}(\tau),\ U_{Hi}(\tau)) = U_{bound} \tag{15}$$

$$U_{Lo}(\tau) = \sum_{\tau_i \in \pi, q=0}^{q<p} U_{i,\ Lo,q} \tag{16}$$

$$U_{Hi}(\tau) = \sum_{\tau_i \in Hi(\pi), q=0}^{q<p} U_{i,\ Hi,q} \tag{17}$$

where π is the task set, and $Hi(\pi)$ is a subset of π that contains only *Hi* criticality tasks.

- $[U_l, U_U]$ Utilizations are generated uniformly from this range, with:
  $0 \leq U_L \leq U_U \leq 1$
  $[Z_l, Z_U]$: Ratio between the Hi-criticality utilization of a task and its Lo-criticality utilization with
  $0 \leq Z_L \leq Z_U$.
- WCET for criticality level Lo:
  $C_{i,Lo,q} = U_{i,Lo,q} \cdot T_i$.
- WCET for criticality level Hi:
  $C_{i,Hi,q} = U_{i,Hi,q} \cdot T_i \ if \ L_i = Hi$. Otherwise, $C_{i,\ Hi,q} = C_{i,\ Lo,q}$.
- The affinity values were assigned using one of the two methods described in the previous section.

6.2. Best Affinity First Performance Evaluation Results

The parameters used to generate the task sets are provided in graph captions. For each plot, one parameter was varied, whereas the others were fixed. The utilization of each processor must satisfy the necessary condition bound on the load metric for any m-processing unit system (9). Each data point is determined by randomly generating 100 task sets.

6.2.1. Proposed MC Task Model vs. Classical MC Task Model

The main differences between our proposed model and the classical MC task model are as follows: a new parameter called affinity is introduced, and the computation time, in the form of worst-case execution time, becomes a bi-dimensional array (with one dimension being the criticality level and one the processing element). Table 1 briefly illustrates the differences.

Table 1. Parameter Comparison Between the Proposed MC Task Model and the Classical MC Task Model.

Parameters	Proposed MC Task Model	Classical MC Task Model
Period	T_i	T_i
Deadline	D_i	D_i
Criticality level	L_i	L_i
WCET	$\left\{C_{i,j,q} \middle\vert j \in 1 \ldots l,\ q \in 1 \ldots p\right\}$	$\left\{C_{i,j} \middle\vert j \in 1 \ldots l\right\}$
Affinity	$\left\{A_{i,q} \middle\vert q \in 1 \ldots p\right\}$	-

Next, we consider an example of an application model using both the proposed and classical models. We also consider a sensor node modeled as a dual-criticality system with three processing elements. The sensor node runs the aforementioned IoT sensing application and consists of four tasks. The affinity score for each task is computed in Table 2 according to the two algorithms presented previously: Algorithms 1 and 2 (because the number of PEs exceeds the number of criticality levels).

Table 2. Task Model Example 1.

Task	Functionality	Proposed MC Task Model						Classical MC Task Model				
		T_i	D_i	L_i	$C_{i,j,q}$	$A_{i,q}$ Algorithm 1	$A_{i,q}$ Algorithm 2	T_i	D_i	L_i	$C_{i,j,q}$	T_i
M_1	sensing	8	8	*Lo*	2 2 1 1 4 4	2 3 1	3 1 2	8	8	*Lo*	4	4
M_2	sensing	17	17	*Hi*	4 7 6 8 3 5	2 1 3	1 3 2	17	17	*Hi*	6	8
M_3	*log*	24	24	*Lo*	5 5 8 8 7 7	3 1 2	3 1 2	24	24	*Lo*	8	8
M_4	communication	42	42	*Hi*	5 10 8 12 13 17	3 2 1	2 3 1	42	42	*Hi*	13	17

The tasks are mapped on processing elements using the following methods: on the one hand, BAF (by computing the affinity score according to Algorithm 1), and BFDU and BFDC using the classical MC task model, considering the worst-case execution time vector (the line with the highest values from the $C_{i,j,q}$ matrix). The task-mapping results are presented in Table 3.

Table 3. Task Mapping Example 1.

BAF (using Algorithm 1)	$\{M_3, M_4\} \in \Psi_1$ $\{M_1\} \in \Psi_2$ $\{M_2\} \in \Psi_3$	**BFDU**	$\{M_1, M_2\} \in \Psi_1$ $\{M_3, M_4\} \in \Psi_2$
BAF (using Algorithm 2)	$\{M_1, M_3\} \in \Psi_1$ $\{M_2, M_4\} \in \Psi_2$	**BFDC**	$\{M_2, M_4\} \in \Psi_1$ $\{M_1, M_3\} \in \Psi_2$

For a system with four criticality levels, {*Lo*, *M1*, *M2*, *Hi*}, and three processing elements, the affinity score is computed in Table 4 according to Algorithms 1 and 3 (because the number of PEs is lower than the number of criticality levels).

Table 4. Task Model Example 2.

Task	Functionality	Proposed MC Task Model						Classical MC Task Model					
		T_i	D_i	L_i	$C_{i,j,q}$	$A_{i,q}$ Algorithm 1	$A_{i,q}$ Algorithm 3	T_i	D_i	L_i	$C_{i,j,q}$	T_i	D_i	L_i
M_1	sensing	15	15	*Lo*	2 2 2 2 1 1 1 1 5 5 5 5	2 3 1	3 2 1	15	15	*Lo*	5	5	5	5
M_2	sensing	20	20	M_1	4 5 5 5 3 6 6 6 6 9 9 9	3 2 1	2 3 1	20	20	M_1	6	9	9	9
M_3	sensing	30	30	*Hi*	5 7 9 12 8 10 13 15 4 8 11 14	3 1 2	2 1 3	30	30	*Hi*	8	10	13	15
M_4	sensing	40	40	M_2	8 14 17 17 9 10 13 13 11 14 16 16	1 3 2	3 2 1	40	40	M_2	11	14	16	16

The tasks were mapped using the same partitioning heuristics as those listed in Table 3. The mapping results are presented in Table 5.

Table 5. Task Mapping Example 2.

BAF (using Algorithm 1)	$\{M_2, M_3\} \in \Psi_1$ $\{M_1, M_4\} \in \Psi_2$	**BFDU**	$\{M_1, M_2, M_3\} \in \Psi_1$ $\{M_4\} \in \Psi_2$
BAF (using Algorithm 3)	$\{M_1, M_4\} \in \Psi_1$ $\{M_2\} \in \Psi_2$ $\{M_3\} \in \Psi_3$	**BFDC**	$\{M_2, M_3, M_4\} \in \Psi_1$ $\{M_1\} \in \Psi_2$

In conclusion, by considering a matrix of computation times instead of a vector, we can model the computation time not only as a function of the criticality level but also by considering the influence of the hardware particularities of the processing elements on the task computation time. Moreover, the introduction of a new parameter called affinity offers a certain degree of versatility to the task model because this parameter can be computed according to any optimization function. Our proposed model offers better resource management by achieving lower total utilization, as discussed in the following paragraphs.

6.2.2. BAF vs. BFDU

In the following paragraphs, we compare the results of different mapping functions in terms of total utilization. To reduce the gap between the two MC task models for all mapping functions, the computation time was considered as a matrix, as in our proposed model. The only difference between the task models used for BAF and BFDU is that BAF also uses the affinity parameter, whereas BFDU and BFDC do not. For the BAF, affinity values were assigned according to the WCET in the Hi mode for Hi-criticality tasks and the WCET in the Lo mode for Lo-criticality tasks.

In Figure 3a, the task set utilization bound (x-axis) ranges from 0.4 to 1.0 times the number of processors in steps of 0.1; in step 3.b, the number of processors (x-axis) ranges from 2 to 12 in steps of 2. In Figure 3c, the number of tasks (x-axis) ranges from 10 to 24 in steps 2 and 3.d the percentage of Hi-criticality tasks (x-axis) was varied, ranging from 0.2 to 0.6 in steps of 0.1. The average total utilization is depicted on the y-axis.

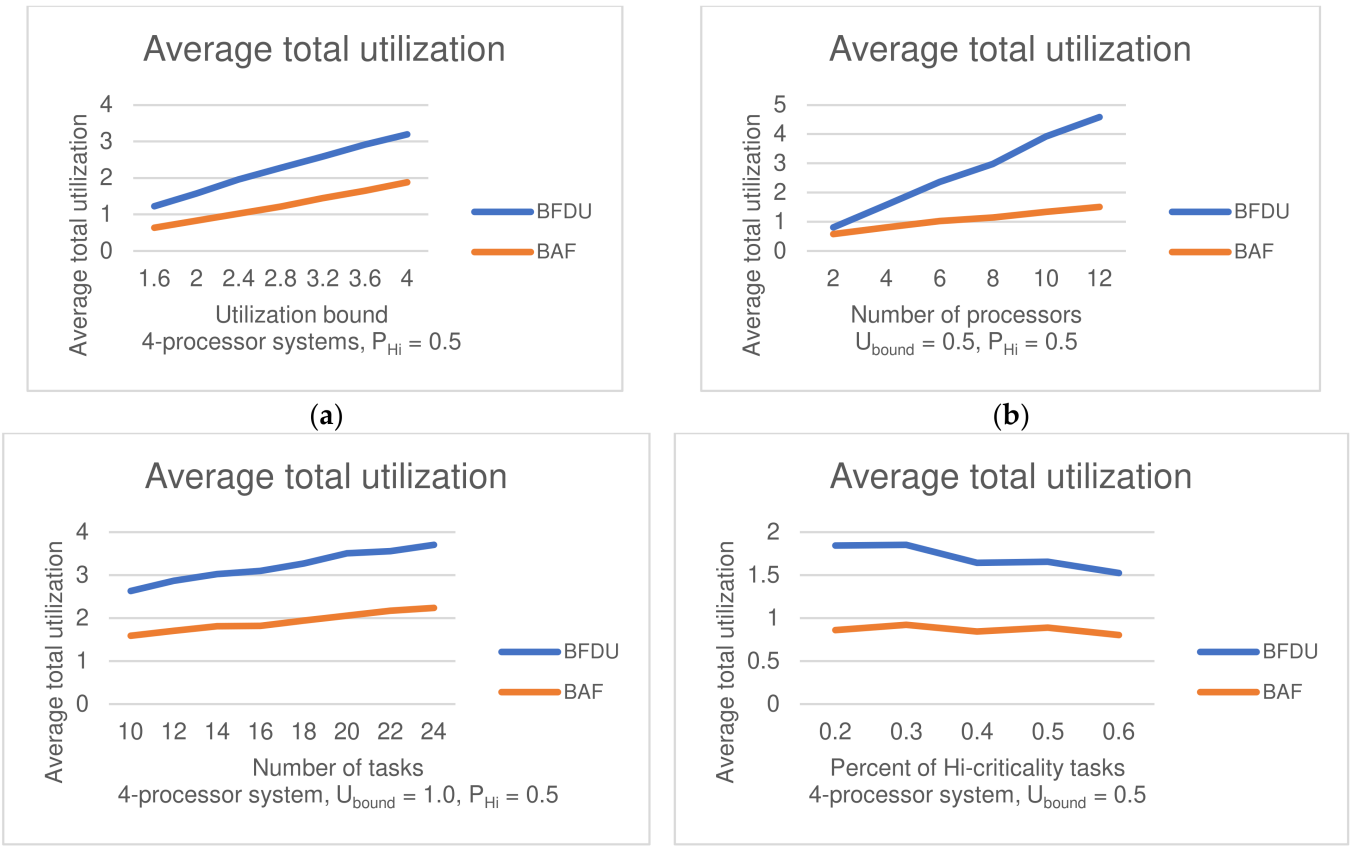

Figure 3. Average total processor utilization by varying: (**a**). the utilization bound, (**b**). the number of processors, (**c**). the number of tasks, (**d**). the percent of Hi-criticality tasks. $U_L = 0.05$, $U_U = 0.75$, $Z_L = 1$, $Z_U = 8$.

From the simulation results presented in Figure 3, we obtain the following results: by varying the number of processors, BAF shows better results with an average value of 1.64, meaning almost 60%, and by varying the number of tasks, there is a reduction in the total utilization when using BAF compared to BFDU with 1.29, meaning about 40%.

6.2.3. BAF vs. BFDC

In this case, affinity values were assigned according to the criticality level of each task. Four experiments were conducted (Figure 4) similar to the first set. Each data point is determined by randomly generating 100 task sets. The average affinity deviations for the task sets used for Figure 4a are depicted in Figure 5 based on Equation (14).

In Figure 4, by varying the number of processors, BAF has better results with an average value of 1.2, meaning about 43%, and by varying the number of tasks, there is a reduction in the total utilization when using BAF compared to BFDC with 0.51, meaning about 15%.

In conclusion, the average total utilization functions for BFDU and BFDC have a similar growth tendency compared to BAF, which is reaching better results with almost a constant factor (between 0.5-1 for BFDU and between 0.2-0.7 for BFDC). The case where the BAF function growth is significantly better is when the number of processors/ processing units is increased.

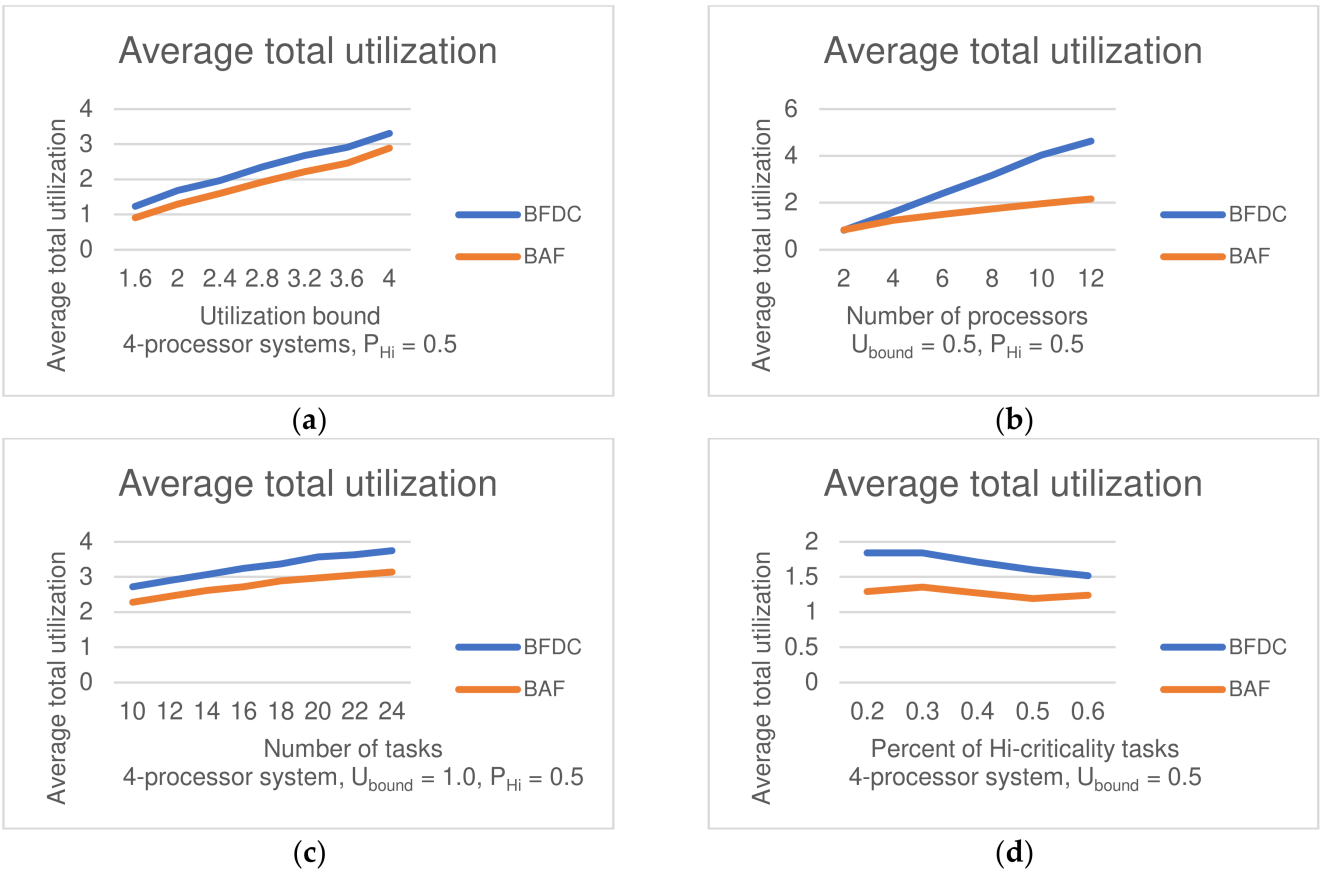

Figure 4. Average total processor utilization by varying: (**a**). the utilization bound, (**b**). the number of processors, (**c**). the number of tasks, (**d**). the percentage of Hi-criticality tasks. $U_L = 0.05$, $U_U = 0.75$, $Z_L = 1$, $Z_U = 8$.

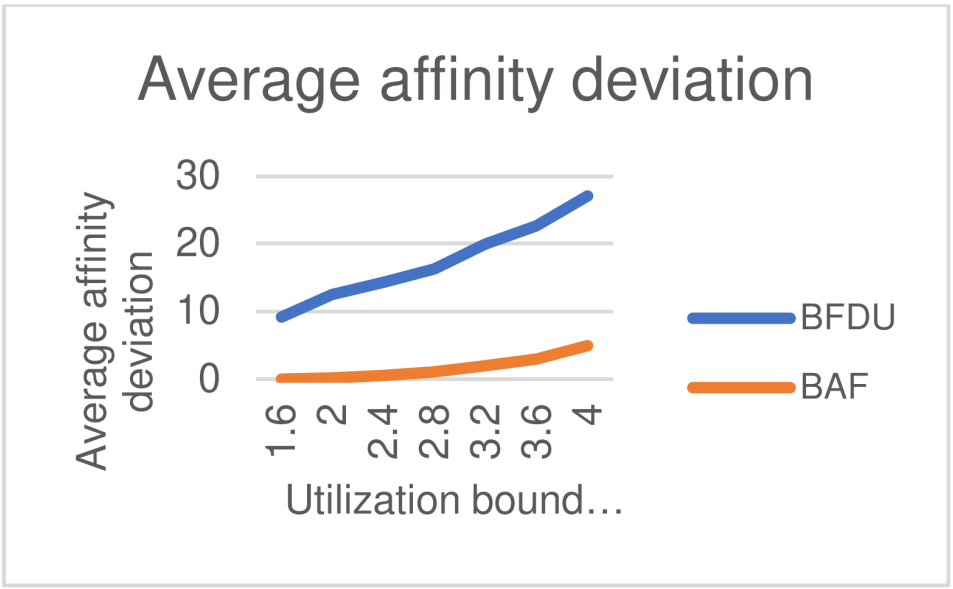

Figure 5. Average affinity deviation by varying the utilization bound. $U_L = 0.05$, $U_U = 0.75$, $Z_L = 1$, $Z_U = 8$.

7. Conclusions and Future Perspectives

With the continuous improvement in IoT technology and increasingly complex applications, some of them with strict timing constraints are envisioned to be developed. Under these circumstances, the implementation of new and effective RT-IoT architectures may offer much-required technological support. From our perspective, combining the underlying mechanisms of mixed-criticality systems with the Internet of Things offers a huge opportunity for the research and development of complex distributed applications.

In this study, we mathematically formalized the MC-IoT concept. We also propose an effective methodology for scheduling mixed-criticality applications on IoT nodes. A novel hardware-aware extension of the classical MCS task model was proposed to support this methodology, along with a new task mapping function: Best Affinity First.

The performance evaluations prove that BAF has significantly better results than the existing BFDU and BFDC task-mapping techniques in terms of total task utilization. The difference in performance increases with the number of processing elements. These results are underlying the suitability of the proposed partitioning algorithm to the IoT scenario where more numerous processing elements are used compared to the classic MCSs.

Moreover, a hardware-aware scheduling approach increases the potential for the development of the distributed heterogeneous systems running real-time mixed-criticality applications such as 6G networks and their applications.

The proposed model can be further extended in order to include other aspects besides the hardware affinity. Depending on the other resources in the system that need to be more efficiently used, other functions for setting the affinity score can also be developed.

Author Contributions: Conceptualization, C.S.S., E.A.C., M.V.M. and D.I.C.; methodology, C.S.S., E.A.C. and V.S.; software, E.A.C.; validation, C.S.S., E.A.C. and V.S.; formal analysis, C.S.S., E.A.C., M.V.M. and D.I.C.; investigation, C.S.S. and E.A.C.; resources, M.V.M. and D.I.C.; writing—original draft preparation, C.S.S. and E.A.C.; writing—review and editing, M.V.M. and D.I.C.; visualization, C.S.S., E.A.C., M.V.M., V.S. and D.I.C.; supervision, M.V.M. and D.I.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

Model Class	Task Model	Authors	Year	Number of Parameters	Main Task Parameters	Criticality Levels
RTS	Periodic	Liu & Layland	1973	2	C—computation time, T—period	-
RTS	DAG	Baruah	1998	2	e—execution requirement, d—deadline	-
RTS	Sporadic	Mok	1983	3	c—computation time, p—period, d-deadline	-
RTS	Periodic	Hamdaoui & Ramanathan	1995	4	c—computation time, d—deadline, m—tolerable deadline misses from total k deadlines	-
RTS	DAG	Stigge et al.	2011	3	e—execution requirement, d—deadline, r—release time	-

Model Class	Task Model	Authors	Year	Number of Parameters	Main Task Parameters	Criticality Levels
MCS	Generic	Vestal	2007	4	C—computation time (function of L), T—period, D—deadline, L—criticality level	4
MCS	Periodic/ Sporadic	Lee & Shin	2017	5	C—computation time (function of L), T—period, D—deadline, L—criticality level, m—tolerable deadline misses	n
MCS	DAG	Huang et al.	2013	4	C—computation time (function of L), T—period, D—deadline, L—criticality level	n
MCS	DAG	Ekberg & Yi	2016	3	e—execution requirement, d—deadline, m—mode of the corresponding job type	4
MCS	Periodic/ Sporadic	Su and Zhu	2013	6	C—computation time (function of L), T—period, D—deadline, L—criticality level, p_max—maximum period, P_er—early release points	2
MCS	Periodic/ Sporadic	Burns	2015	4	C—computation time (function of L), T—period, D—deadline, L—criticality level	2
MCS	Periodic/ Sporadic	Burns & Baruah	2013	4	C—computation time (function of L), T—period, D—deadline, L—criticality level	2
MCS	Periodic/ Sporadic	Li and He	2017	6	L—criticality level, E—execution time (function of L), M—memory access time (function of L), C—computation time (function of L), T—period, D—deadline	2
MCS	Periodic/ Sporadic	Baruah & Chattopadhyay	2013	4	C—computation time (function of L), T—period (function of L), D—deadline, L—criticality level	2
MCS	Distributed	Wang et al.	2021	5	C—computation time (function of L), T—period, D—deadline, L—criticality level, V—value of task (function of L)	2

For uniformity reasons, the criticality level is represented by L, even if in the original articles there might be other symbols used for it.

Appendix B

Abbreviation	Description
RT-IoT	Real-Time Internet of Things
MC-IoT	Mixed Criticality Internet of Things
MCS	Mixed Criticality System
E-MC	Elastic Mixed Criticality
MS-DTR	Mode-Switching Diagraph Real-Time
ICG	Interference Constraint Graph
WCET	Worst Case Execution Time
PE	Processing Element
BAF	Best Affinity Fit
BFDU	Best Fit Decreasing Utilization
BFDC	Best Fit Decreasing Criticality
Symbol	**Description**
τ_i	Task i
T_i	Period of task i
D_i	Deadline of task i
L_i	Criticality of task i
$C_{i,j}$	Computation time in terms of WCET of task i for criticality level j
$J_{i.k}$	Job k of task i
$a_{i,k}$	Arrival time of job k of task i
$c_{i.k}$	Computation time of job k of task i
$C_{i,j,q}$	Computation time in terms of WCET of task i for criticality level j on processing element q
$A_{i,q}$	Affinity of task i for processing element q
M	Mapping function
Ψ_q	Subset of tasks running on processing element q
R_i	Response time of task i
I_i	Inference or higher priority tasks for task i
$R_{i,k,q}$	Response time of job k, of task i on processing element q
$U_{L_j \Psi_q}$	Processor utilization of the Ψ_q subset of tasks of level L_j on processing element q
$U_{i,\ Lo,q}$	Processor utilization of task i for criticality level Lo on processor q
$U_{i,\ Hi,q}$	Processor utilization of task i for criticality level Hi on processor q
$U_{i,j,q}$	Processor utilization of task i for criticality level j on processor q
$A_{d\tau}$	Total affinity score deviation
U_{bound}	Processor utilization bound

References

1. Velasquez, K.; Abreu, D.P.; Assis, M.R.M.; Senna, C.; Aranha, D.F.; Bittencourt, L.F.; Laranjeiro, N.; Curado, M.; Vieira, M.; Monteiro, E.; et al. Fog orchestration for the Internet of Everything: State-of-the-art and research challenges. *J. Internet Serv. Appl.* **2018**, *9*, 14. [CrossRef]
2. Viel, F.; Silva, L.A.; Leithardt, V.R.Q.; Santana, J.F.D.P.; Teive, R.C.G.; Zeferino, C.A. An Efficient Interface for the Integration of IoT Devices with Smart Grids. *Sensors* **2020**, *20*, 2849. [CrossRef] [PubMed]
3. Chen, C.Y.; Hasan, M.; Mohan, S. Securing real-time internet-of-things. *Sensors* **2018**, *18*, 4356. [CrossRef] [PubMed]
4. Tămaş–Selicean, D.; Pop, P.; Steiner, W. Design optimization of TTEthernet-based distributed real-time systems. *Real-Time Syst.* **2015**, *51*, 1–35. [CrossRef]
5. Calvaresi, D.; Marinoni, M.; Sturm, A.; Schumacher, M.; Buttazzo, G. The challenge of real-time multi-agent systems for enabling IoT and CPS. In Proceedings of the International Conference on Web Intelligence, Leipzig, Germany, 23–26 August 2017; ACM: New York, NY, USA; pp. 356–364.
6. Carpenter, T.; Hatcliff, J.; Vasserman, E.Y. A reference separation architecture for mixed-criticality medical and IoT devices. In Proceedings of the 1st ACM Workshop on the Internet of Safe Things, Delft, The Netherlands, 5 November 2017; ACM: New York, NY, USA; pp. 14–19.
7. Moratelli, C.; Johann, S.; Neves, M.; Hessel, F. Embedded virtualization for the design of secure IoT applications. In Proceedings of the 27th International Symposium on Rapid System Prototyping: Shortening the Path from Specification to Prototype, 6–7 October 2016; ACM: New York, NY, USA; pp. 2–6.
8. Yang, Y.; Wang, K.; Zhang, G.; Chen, X.; Luo, X.; Zhou, M.T. MEETS: Maximal energy-efficient task scheduling in homogeneous fog networks. *IEEE Internet Things J.* **2018**, *5*, 4076–4087. [CrossRef]
9. Kamienski, C.; Jentsch, M.; Eisenhauer, M.; Kiljander, J.; Ferrera, E.; Rosengren, P.; Thestrup, J.; Souto, E.; Andrade, W.S.; Sadok, D. Application development for the Internet of Things: A context-aware mixed-criticality systems development platform. *Comput. Commun.* **2017**, *104*, 1–16. [CrossRef]
10. Yao, S.; Hao, Y.; Zhao, Y.; Shao, H.; Liu, D.; Liu, S.; Wang, T.; Li, J.; Abdelzaher, T. Scheduling real-time deep learning services as imprecise computations. In Proceedings of the 2020 IEEE 26th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), Gangnueng, Korea, 19–21 August 2020; pp. 1–10.
11. Jin, H.; Su, L.; Xiao, H.; Nahrstedt, K. Incentive mechanism for privacy-aware data aggregation in mobile crowd sensing systems. *IEEE/ACM Trans. Netw.* **2018**, *26*, 2019–2032. [CrossRef]
12. Zuo, P.; Hua, Y.; Liu, X.; Feng, D.; Xia, W.; Cao, S.; Wu, J.; Sun, Y.; Guo, Y. BEES: Bandwidth-and energy-efficient image sharing for real-time situation awareness. In Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), Atlanta, GA, USA, 5–8 June 2017; pp. 1510–1520.
13. Steinbaeck, J.; Tengg, A.; Holweg, G.; Druml, N. A 3D time-of-flight mixed-criticality system for environment perception. In Proceedings of the 2017 Euromicro Conference on Digital System Design (DSD), Vienna, Austria, 30 August–1 September 2017; pp. 368–374.
14. Dimopoulos, A.C.; Bravos, G.; Dimitrakopoulos, G.; Nikolaidou, M.; Nikolopoulos, V.; Anagnostopoulos, D. A multi-core context-aware management architecture for mixed-criticality smart building applications. In Proceedings of the 2016 11th System of Systems Engineering Conference (SoSE), Kongsberg, Norway, 12–16 June 2016; pp. 1–6.
15. Bravos, G.; Dimitrakopoulos, G.; Anagnostopoulos, D.; Nikolaidou, M.; Kotronis, C.; Politi, E.; Amira, A.; Bensaali, F. Embedded Intelligence in IoT-Based Mixed-Criticality Connected Healthcare Applications: Requirements, Research Achievements and Challenges. *Preprints* **2018**, 2018100216. [CrossRef]
16. Capota, E.A.; Stangaciu, C.S.; Micea, M.V.; Curiac, D.I. Towards Mixed Criticality Task Scheduling in Cyber Physical Systems: Challenges and Perspectives. *J. Syst. Softw.* **2019**, *156*, 204–216. [CrossRef]
17. Gaur, P.; Tahiliani, M.P. Operating systems for IoT devices: A critical survey. In Proceedings of the 2015 IEEE Region 10 Symposium, Ahmedabad, India, 13–15 May 2015; pp. 33–36.
18. Kim, J.E.; Abdelzaher, T.; Sha, L.; Bar-Noy, A.; Hobbs, R.; Dron, W. On maximizing quality of information for the internet of things: A real-time scheduling perspective. In Proceedings of the 2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), Daegu, Korea, 17–19 August 2016; pp. 202–211.
19. Zhang, T.; Gong, T.; Gu, C.; Ji, H.; Han, S.; Deng, Q.; Hu, X.S. Distributed dynamic packet scheduling for handling disturbances in real-time wireless networks. In Proceedings of the 2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Pittsburgh, PA, USA, 18–21 April 2017; pp. 261–272.
20. Sukumaran Nair, A.; Colaco, L.M.; Raveendran, B.; Punnekkat, S. TaskMUSTER: A comprehensive analysis of task parameters for mixed criticality automotive systems. *Sādhanā* **2022**, *47*, 1–23. [CrossRef]
21. Buttazzo, G.C. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2011; Volume 24.
22. Capota, E.A.; Stangaciu, C.S.; Micea, M.V.; Curiac, D.I. Towards Fully Jitterless Applications: Periodic Scheduling in Multiprocessor MCSs Using a Table-Driven Approach. *Appl. Sci.* **2020**, *10*, 6702. [CrossRef]
23. Jeffay, K.; Stanat, D.F.; Martel, C.U. On non-preemptive scheduling of periodic and sporadic tasks. In Proceedings of the Twelfth Real-Time Systems Symposium, San Antonio, TX, USA, 4–6 December 1991; pp. 129–139.

24. Mok, A.K.L. Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment. Diploma Thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1983.
25. Vestal, S. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In Proceedings of the 28th IEEE International Real-Time Systems Symposium (RTSS 2007), Tucson, AZ, USA, 3–6 December 2007; pp. 239–243.
26. Baruah, S.; Li, H.; Stougie, L. Towards the design of certifiable mixed-criticality systems. In Proceedings of the 2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium, Stockholm, Sweden, 12–15 April 2010; pp. 13–22.
27. Zeng, L.; Xu, C.; Li, R. Partition and Scheduling of the Mixed-Criticality Tasks Based on Probability. *IEEE Access* **2019**, *7*, 87837–87848. [CrossRef]
28. Burns, A. An augmented model for mixed criticality. In *Mixed Criticality on Multicore/Manycore Platforms (Dagstuhl Seminar 15121)*; Baruah, S.K., Cucu-Grosjean, L., Davis, R.I., Maiza, C., Eds.; Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik: Dagstuhl, Germany, 2015; Volume 5.
29. Baruah, S.; Guo, Z. Mixed-criticality job models: A comparison. In Proceedings of the Workshop on Mixed-Criticality Systems (WMC'15), San Antonio, TX, USA, 1 December 2015.
30. Burns, A.; Baruah, S. Towards a more practical model for mixed criticality systems. In Proceedings of the Workshop on Mixed-Criticality Systems (Colocated with RTSS), Berlin, Germany, 1–4 December 2020.
31. Baruah, S.; Chattopadhyay, B. Response-time analysis of mixed criticality systems with pessimistic frequency specification. In Proceedings of the 2013 IEEE 19th International Conference on Embedded and Real-Time Computing Systems and Applications, Taipei, Taiwan, 19–21 August 2013; pp. 237–246.
32. Li, Z.; He, S. Fixed-priority scheduling for two-phase mixed-criticality systems. *ACM Trans. Embed. Comput. Syst.* **2017**, *17*, 1–20. [CrossRef]
33. Wang, W.; Mao, C.; Zhao, S.; Cao, Y.; Yi, Y.; Chen, S.; Liu, Q. A smart semipartitioned real-time scheduling strategy for mixed-criticality systems in 6G-based edge computing. *Wirel. Commun. Mob. Comput.* **2021**, *2021*, 1–11. [CrossRef]
34. Su, H.; Zhu, D. An elastic mixed-criticality task model and its scheduling algorithm. In Proceedings of the Conference on Design, Automation and Test in Europe, Grenoble, France, 18–22 March 2013; pp. 147–152.
35. Lee, J.; Shin, K.G. Development and use of a new task model for cyber-physical systems: A real-time scheduling perspective. *J. Syst. Softw.* **2017**, *126*, 45–56. [CrossRef]
36. Hamdaoui, M.; Ramanathan, P. A dynamic priority assignment technique for streams with (m, k)-firm deadlines. *IEEE Trans. Comput.* **1995**, *44*, 1443–1451. [CrossRef]
37. Baruah, S.K. A general model for recurring real-time tasks. In Proceedings of the 19th IEEE Real-Time Systems Symposium (Cat. No. 98CB36279), Madrid, Spain, 4 December 1998; pp. 114–122.
38. Stigge, M.; Ekberg, P.; Guan, N.; Yi, W. The digraph real-time task model. In Proceedings of the 2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium, Chicago, IL, USA, 11–14 April 2011; pp. 71–80.
39. Ekberg, P.; Yi, W. Schedulability analysis of a graph-based task model for mixed-criticality systems. *Real-Time Syst.* **2016**, *52*, 1–37. [CrossRef]
40. Huang, P.; Kumar, P.; Stoimenov, N.; Thiele, L. Interference Constraint Graph—A new specification for mixed-criticality systems. In Proceedings of the 2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA), Cagliari, Italy, 10–13 September 2013; pp. 1–8.
41. Alimi, I.A.; Patel, R.K.; Zaouga, A.; Muga, N.J.; Xin, Q.; Pinto, A.N.; Monteiro, P.P. Trends in Cloud Computing Paradigms: Fundamental Issues, Recent Advances, and Research Directions toward 6G Fog Networks. In *Moving Broadband Mobile Communications Forward: Intelligent Technologies for 5G and Beyond*; IntechOpen: London, UK, 2021; p. 3.
42. Kelly, O.R.; Aydin, H.; Zhao, B. On partitioned scheduling of fixed-priority mixed-criticality task sets. In Proceedings of the 2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications, Changsha, China, 16–18 November 2011; pp. 1051–1059.
43. Santy, F.; George, L.; Thierry, P.; Goossens, J. Relaxing mixed-criticality scheduling strictness for task sets scheduled with fp. In Proceedings of the 2012 24th Euromicro Conference on Real-Time Systems, Pisa, Italy, 11–13 July 2012; pp. 155–165.
44. Socci, D. Scheduling of Certifiable Mixed-Criticality Systems. Diploma Thesis, L' Ecole Doctorale Mathematiques, Sciences et Technologies de l'Information, Informatique Universite de Grenoble Alpes, Grenoble, France, 2016.
45. Lupu, I.; Courbin, P.; George, L.; Goossens, J. Multi-criteria evaluation of partitioning schemes for real-time systems. In Proceedings of the 2010 IEEE 15th Conference on Emerging Technologies & Factory Automation (ETFA 2010), Bilbao, Spain, 13–16 September 2010; pp. 1–8.
46. Li, H.; Baruah, S. Outstanding paper award: Global mixed-criticality scheduling on multiprocessors. In Proceedings of the 2012 24th Euromicro Conference on Real-Time Systems, Pisa, Italy, 11–13 July 2012; pp. 166–175.
47. Guan, N.; Ekberg, P.; Stigge, M.; Yi, W. *Improving the Scheduling of Certifiable Mixed-Criticality Sporadic Task Systems*; Technical Report 2013-008; Uppsala University: Uppsala, Sweden, 2013; pp. 1–12.