SCHEDULING METHODS FOR MIXED CRITICALITY REAL-TIME DISTRIBUTED SYSTEMS

PhD Thesis - Abstract

For obtaining the Scientific Title of PhD in Engineering from
Politehnica University of Timişoara
in the field of Computer and Information Technology

author ing. Eugenia Ana CAPOTA
thesis supervisor Prof.univ.dr.habil.ing. Mihai V. MICEA

November 2021

Contents:

1. Introduction

1.1 Main research domains

Real-time systems (RT) are present in our day-to-day life throughout numerous domains. These systems involve a real-time response due to the direct interaction with the environment but also efficient resource management.

A concept which presents high interest in classical real-time systems is to combine several critical functionalities onto the same platform. This has led to the development of a new class of systems: mixed criticality systems (MCSs). The MCS concept basically refers to "*an embedded computing platform in which application functions of different criticality share computation and/or communication resources*" [1].

Many real-time critical applications have already been implemented using distributed architectures [2]. These architectures contain a collection of independent components, that run on multiple processing units and communicate with each other through a network [3].

A vast domain which uses distributed architectures is represented by cyber physical systems (CPSs). These systems include *„a logic function in their design such that either the*

state of the logic function could be altered by a change in the state of the real world, or the state of the real world could be modified by a change in the state of the logic function, or both" [4].

Another domain that extends the applicability of distributed heterogenous systems is Internet of Things (IoT). In an era of telecommunication and interconnectivity, IoT is a promising new technologies which can connect the intelligent objects that surround us, creating a network, often distributed over large geographical regions [5].

This thesis focuses on scheduling methods for mixed criticality systems.

1.2 Thesis objectives

The main goal of this thesis is to develop a standardized task set execution model for distributed real-time mixed criticality systems which can take into account the hardware features of the platform on which it runs, but also to introduce a perfectly periodical algorithm for mixed criticality systems with multiple processing units.

Thus, the research carried out can be summarized into four objectives, each divided into several stages:

[O1]. An overview of the state-of-the-art mixed criticality systems, with emphasis on distributed platforms by:
- *[T1.1]. Comparing different task execution models from the literature.*
- *[T1.2]. Classifying the main scheduling algorithms by the platform on which they run.*

[O2]. The implementation of a task execution model by:
- *[T2.1]. Defining a task set execution model for distributed real-time mixed criticality systems.*
- *[T2.2]. Implementing the task model onto a simulation environment.*
- *[T2.3]. Testing the previously proposed task model, on the simulator, using scheduling algorithms.*

[O3]. The development of a scheduling mechanism by:
- *[T3.1]. Implementing a scheduling algorithm for real-time mixed criticality systems.*
- *[T3.2]. Designing a schedulability test for the algorithm.*

[O4]. The testing and validation of the algorithm by:
- *[T4.1]. Testing and comparing different versions of the algorithm developed previously with the help of a simulation environment.*

1.3 Thesis structure

Chapter 1 contains an introduction to real-time MCSs, the main objectives and the thesis structure.

Chapter 2 presents the evolution of task execution models and a classification of the scheduling algorithms for MCSs from the literature.

Chapter 3 is divided into two sections: the first one discusses CPSs, and the second one IoT platforms. The previously classified algorithms are compared based on a set of common CPSs attributes. The challenges and advantages of integrating MCSs into CPSs are also highlighted. The next section addresses IoT by proposing a MC-IoT architecture for real-time distributed MCSs.

Chapter 4 defines a new task execution model for distributed MCSs which allows for an efficient administration of the platform resources. Next, a task mapping methodology is introduced which takes both temporal and hardware requirements into account. The methodology uses different techniques to determine the affinity score of a task for each processing element.

Chapter 5 describes the simulation environment adapted for heterogeneous distributed

MCSs.

Chapter 6 evaluates the two affinity assignment strategies and the task mapping algorithm by implementing and testing them on the simulation environment.

Chapter 7 presents the proposed algorithm for MCSs with a single processing unit. The scheduling method is called FENP_MC (Fixed Execution Non-Preemptive Mixed Criticality) and is a real-time, table-driven, non-preemptive algorithm adapted for MCSs, based on the FENP (Fixed Execution Non-Preemptive) technique for classical real-time systems, which guarantees a perfectly periodical (jitterless) execution of tasks in a time-triggered environment.

Chapter 8 proposes a scheduling method by implementing a task partitioning heuristic for homogenous systems with multiple processing units, namely P_FENP_MC (Partitioned Fixed Execution Non-Preemptive Mixed Criticality).

Chapter 9 analyzes the performance of the proposed algorithm by comparing it with other scheduling methods, in a non-preemptive context, in terms of success ratio and jitter value.

Chapter 10 summarizes the main contributions of this thesis and presents some future perspectives.

2. State-of-the-art: Mixed Criticality Systems

In MCSs, as in real-time systems, the basic level of an application is represented by the task. A system criticality mode switch will result in the dropping of low criticality tasks, while high criticality tasks will continue to execute according to their high criticality parameters. The first task execution model for MCSs was proposed by Vestal [6]:

$$\tau_i = \left\{T_i, D_i, L_i, \left\{C_{i,L_j} \mid j \in 1 \dots l\right\}\right\} \tag{2-1}$$

where:

- l is the number of criticality levels;
- T_i represents the (minimum) arrival interval between two consecutive jobs of the same task i;
- D_i indicates the time by which any job execution needs to complete, relatively to its release time;
- L_i is the criticality level;
- C_{i,L_j} represents the worst-case execution time, WCET (vector of values – one per criticality level, for levels less than or equal to the criticality level L_i, expressing the worst-case execution time for each criticality level).

Numerous scheduling algorithms have been developed, based on the classical task model introduced by Vestal, for single and multiple processing units in MCSs. Regarding the algorithms running on a single processing unit or at the unit level in a system with multiple processing units, they can be classified according to the way the priority is assigned to the task instances (jobs) [7]:

- Fixed Task-Priority (FTP) class – All the jobs generated by a given task are assigned the same priority.
- Fixed Job-Priority (FJP) class – Different jobs of the same task may have different priorities. However, the priority of each job may not change between its arrival time and its completion time.
- Dynamic Priority (DP) class – Priorities of jobs may change between their release times and their completion times.

- Hybrid Priority (HP) class – The scheduling policies incorporate features of multiple scheduling algorithm classes.

For systems with multiple processing units, scheduling algorithms can be classified into four categories [3, 7-9]:

- Class P: Partitioned schedulers – each task is assigned to a single processing unit.
- Class G: Global schedulers – tasks can migrate from one processing unit to another.
- Class C: Clustered/semi-partitioned schedulers – hybrid approach between the partitioned and global schedulers which refers to a group of processing units where each cluster is divided into sub-clusters.
- Class D: Distributed schedulers – they use distributed middleware in order to interconnect partitions. A partition can have one or multiple processing units.

3. Distributed systems

3.1 Cyber physical systems

CPSs applicability extends to fields such as: automotive, avionics, medical devices, industrial platforms, etc. Even though CPSs include a wide range of systems, very different from an architectural and functional perspective, one can identify a set of common attributes [10]:

- **Heterogeneity** – it refers to different hardware specifications, various application and power consumption requirements.
- **Power management** – must be considered, especially for hardware components powered using batteries.
- **Dynamism and self-adaptability** – due to the interaction with the physical environment, which is often dynamic and unpredictable, the system must be able to face changes in a real-time manner.
- **Robustness** – since the environment in which CPSs operate is usually subject to several uncertainties in terms of run-time behavior, it is also desirable that critical applications are not affected by the failures or computation overload caused by any other application.
- **Distribution** – refers to scheduling algorithms developed for systems with multiple processing units, which consider the location of components and the communication between them.
- **Scalability** – when considering systems with multiple processing units, their scalability in terms of hardware components, application design, analysis, coding and testing is of relatively great concern. Therefore, scalability of the task models, scheduling mechanisms and of the scheduling analysis must also be provided.
- **Security and isolation** – security is difficult to ensure in a platform with different criticality levels. In a complex system, containing different functionalities, critical components must be isolated from the non-critical ones in terms of temporal behavior and resource usage.

According to the previously mentioned attributes, a series of compliance levels are presented in this thesis (Table 1) which can be used to evaluate scheduling algorithms for mixed criticality systems.

Table 1. Levels of compliance.

Attribute	Levels	Level description
Heterogeneity	At task level	Handles different task set types
	At device level	Handles different architectures
Power management	-	Does not consider power management

	Low	Considers static power consumption
	Moderate	Considers dynamic power consumption
	High	Considers static and dynamic power consumption
Dynamism and self-adaptability	Low	Does not accept dynamic task loading
	Moderate	Accepts limited dynamic task loading
	High	Task sets can be loaded dynamically during run time
Robustness	Low	Does not handle overload
	Moderate	Handles overload only for high criticality levels
	High	Handles overload for both low and high criticality levels
Distribution	Yes/No	Algorithms for distributed systems/Other algorithms
Scalability	Yes/No	Scalable with respect to the number of criticality levels/Not scalable
Security and isolation	Low	Only temporal isolation between criticality levels
	Moderate	Temporal and spatial isolation only for high criticality levels
	High	Temporal and spatial isolation between different criticality levels

3.2 Internet of Things

The IoT concept has been applied on a large scale in different domains, such as medical [11] or industrial systems [12, 13]. Therefore, different types of physical IoT architectures have been proposed and described in the literature. The Fog based architecture is currently the most suitable for complex distributed real-time applications. Fog based architectures are comprised of three levels [14]: Cloud (here data is stored in data centers and delivered as service to users over the Internet), Fog (where data is stored and processed locally, close to the end users, in order to eliminate network delays) and Edge (a network of heterogenous and distributed devices).

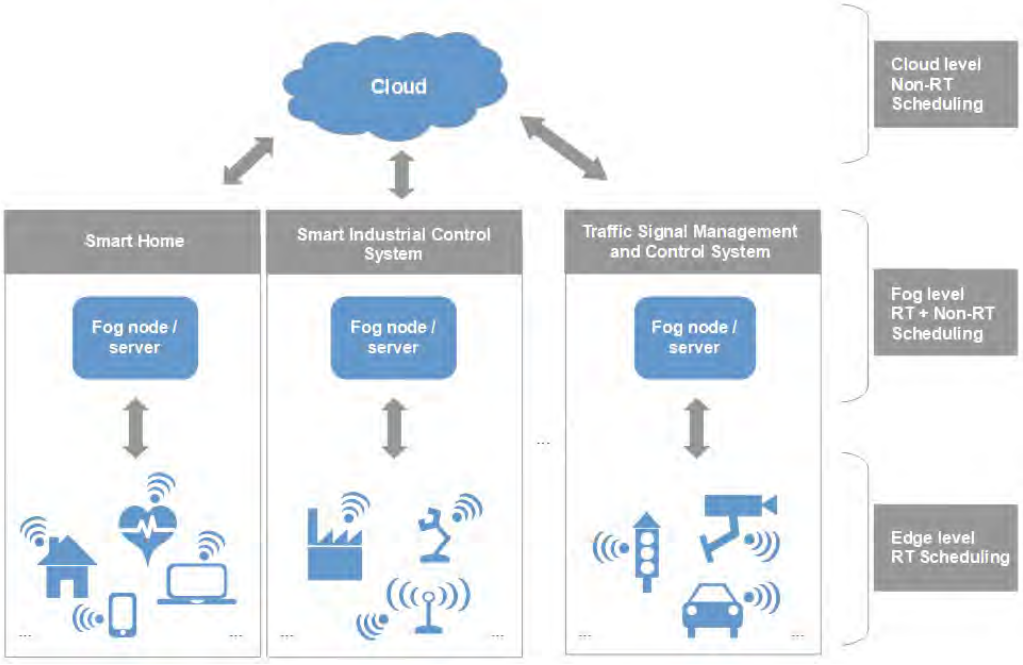

Figure 1. Proposed Fog MC-IoT architecture.

Figure 1 considers one or more centralized heterogenous Fog networks connected to the Cloud. Each network contains a Fog node which stores and processes data from local Edge devices. The Edge nodes are able to communicate with each other and with the Fog node and can be any type of processing capable devices.

The methods used for scheduling tasks vary according to the architecture level. While

there is no time-sensitive communication protocol between Cloud and Edge, the Fog level offers some functionalities which allow real-time scheduling of tasks.

The Edge nodes can run real-time applications divided into mixed criticality tasks, which results in a mixed criticality Edge network. Tasks will be scheduled on each processing element (PE). In this architecture a processing element is represented by a system with one processing unit.

Based on the proposed architecture, a task execution model is introduced and a mathematical formalization of the scheduling problem is done.

Adapting the concept of MCSs to IoT architectures gives birth to a new scheduling paradigm called MC-IoT, for which this thesis proposes the following definition:

Definition 1. *Mixed Criticality-Internet of Things (MC-IoT) are systems running real-time tasks of different criticalities at the Edge level of IoT architectures.*

The scheduling problem at the Edge level can be divided in two sub problems:

1) Task mapping (at the Fog level) – each task must be allocated/mapped on a processing element.
2) Local task scheduling (at the Edge device level) – all the tasks mapped on a processing element must be schedulable.

4. Proposed mapping technique

This thesis starts from the classical MCSs task model introduced by Vestal in [6] and proposes an extension consisting of a new parameter: the affinity score. A_i is defined as a vector of values, one per processing element, representing the affinity of task i for each processing element of the system. The affinity score is an integer between 0 and p, where p is the number of processing elements (PE). A higher value means a higher affinity, while 0 means no affinity.

Therefore, the computation time will not only be expressed as a function of the criticality level (L_i), but also as a function of the processing element on which the task executes (PE_q). The task model (2-1) becomes:

$$\tau_i = \left\{T_i, D_i, L_i, \left\{C_{i,L_j\,PE_q} \mid j \in 1 \dots l, q \in 1 \dots p\right\}, \left\{A_{i,PE_q} \mid q \in 1 \dots p\right\}\right\} \qquad (4\text{-}1)$$

where:

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

The affinity score can be set statically by the task creator, or computed using an algorithm based on the resources needed by the task and on the task computation time on each processing element.

Using the task model previously defined and the scheduling problem formalized in Section 3.2, this thesis proposes a methodology for mapping tasks on distributed mixed criticality systems. The methodology comprises of different methods for setting the newly introduced task parameter, namely the affinity score and for defining a suitable mapping function in order to respect both application and resource constraints. A dual-criticality system is considered (Lo – low criticality and Hi – high criticality), but the algorithm can be implemented on platforms with more than two criticality levels.

4.2 Set affinity score based on computation time:

Step 1: Extract the task computation time for the highest criticality level on each PE

(the second column in the generated computation time matrix C_{i,L_jPE_q}), by copying it into an array of structures X_{i,L_2PE_q}. Each structure has a computation time value X_{i,L_2PE_q} and a PE index (q), where i represent the task number and it is fixed and L_2 represents the highest criticality level. Index q varies from 1 to p.

Step 2: Extract the matrix line index of the maximum value for X_{i,L_2PE_q} from the array, where q varies from 1 to p.

Step 3: In the affinity array, set the affinity score A_{i,PE_q} to 1 for the PE corresponding to the highest X_{i,L_2PE_q} value, and then 2 for the PE corresponding to the highest X_{i,L_2PE_q} from the array after setting the element with the highest value from the first iteration $X_{i,L_2PE_{index}}$ to 0, 3 for the remaining highest value and so on, while $q \leq p$.

4.3 Set affinity score based on criticality level:

If the number of processing elements exceeds the number of criticality levels:

Step 1: Build an array of structures PE_{q,L_j}. Each structure has a criticality level L_j, which represents the expected criticality level of the tasks to be partitioned on PE_q and a PE index (q). Index q varies from 1 to p, while L_j is given by computing the following $L_j = PE_q \ mod \ l$, where l is the number of criticality levels and mod represents the modulo operation. If L_j is 0, then the algorithm considers $L_j = l$.

Step 2: Assign each task τ_i according to the criticality level L_i. There will be two subsets: PEs with expected criticality PE_{q,L_j} equal to L_i and PEs with expected criticality PE_{q,L_j} not equal to L_i. The affinity score A_{i,PE_q} will have the highest values for the first subset of PEs. For each subset of PEs, the affinity score is set according to the computation time.

If the number of criticality levels exceeds the number of processing elements:

Step 1: For each task τ_i, get the PE on which it is expected to run by computing the following $PE_q = L_i \ mod \ p$, where p is the number of PEs and mod represents the modulo operation. If PE_q is 0, then the algorithm considers $PE_q = p$. Next, set the affinity score A_{i,PE_q} for PE_q to p.

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

4.4 Task mapping

This thesis introduces a new mapping algorithm, namely Best Affinity Fit (BAF), which partitions tasks to processing elements according to the affinity score.

For each task τ_i:

Step 1: Firstly, it is assumed that task i can be assigned to a PE, therefore a variable called $assign$ is set to 1. Next, the PE with the highest affinity score value for task i is identified, which has a processor utilization less than or equal to 1 for all criticality modes of the system.

Step 2: If a certain PE_{index} does not have enough space for task i, set the affinity score $A_{i,PE_{index}}$ to 0. If there is no PE that can host task i, set $assign$ to 0.

Step 3: If a PE_{index} is found for task i, then add the task to subset Ψ_{index} and update the PE utilization.

5. Simulation environment

The mapping algorithm Best Affinity Fit (BAF) was initially implemented in MATLAB and tested by comparing it with two methods: Best Fit Decreasing Utilization (BFDU) and Best

Fit Decreasing Criticality (BFDC).

Additionally, the task model and mapping algorithm was successfully integrated into a simulation environment [15] developed in C++ for homogeneous mixed criticality systems. Therefore, the simulation environment can also be used for heterogeneous distributed mixed criticality systems (Figure 2). All the task sets were generated randomly in MATLAB using the task set generation algorithm introduced in [16], which is a slight modification of the workload generation algorithm presented by Guan [17]. The graphical interface for viewing task scheduling was also developed in MATLAB.

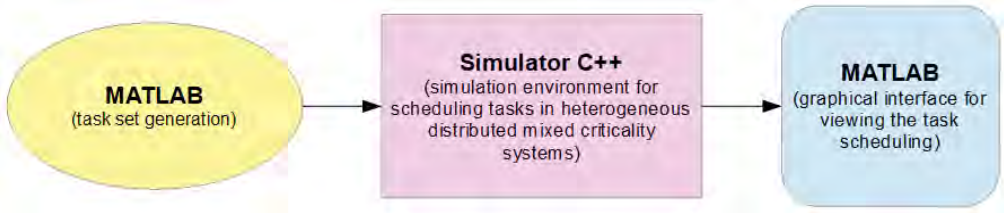

Figure 2. Simulation environment for heterogeneous distributed mixed criticality systems.

6. Performance evaluation

A series of simulation experiments were conducted to evaluate the effectiveness of the new mapping heuristic, Best Affinity Fit (BAF). It has been compared against two other relevant mapping techniques, Best Fit Decreasing Utilization (BFDU) and Best Fit Decreasing Criticality (BFDC), which are some of the most frequently used algorithms in literature [18-20]. BAF dispatches tasks to processors according to the affinity value, while for BFDU and BFDC tasks are first ordered by decreasing utilization, respectively, by decreasing criticality, and then assigned to each processor; the processors are also ordered by their decreasing utilization. Additionally, two affinity assignment strategies were evaluated: one allocates the values according to the WCET, while the other according to the criticality level. A system with two criticality levels was considered $\{Lo, Hi\}$. Each data-point was determined by randomly generating 1000 task sets.

BAF vs. BFDU: Affinity values were assigned according to the WCET in Hi mode for Hi-criticality tasks and the WCET in Lo mode for Lo-criticality tasks.

BAF vs. BFDC: In this case, affinity values were assigned according to the criticality level of each task (Figure 3).

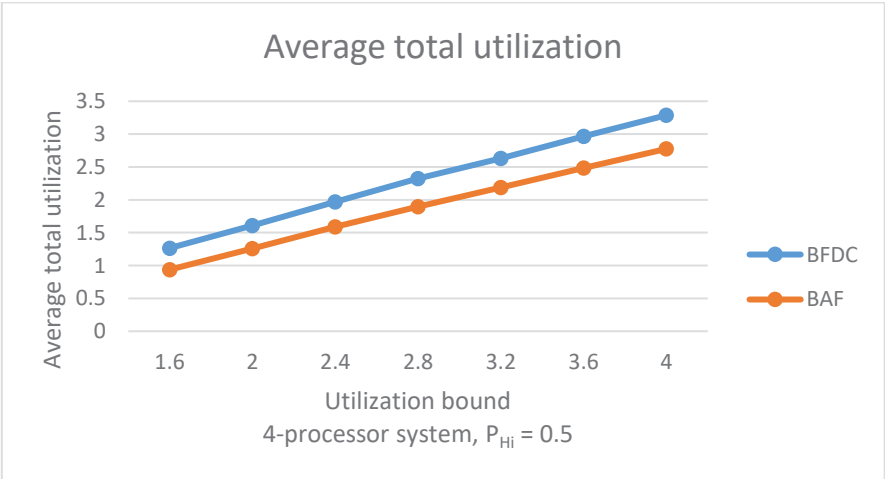

Figure 3. Average total processor utilization by varying the utilization bound. $\boldsymbol{U_L = 0.05, U_U = 0.75, Z_L = 1, Z_U = 8}$.

For both sets of experiments, Best Affinity Fit performed better than BFDU and BFDC with respect to the average total utilization value.

Table 2 depicts the average affinity deviation value for the task sets generated in Figure 3.

Table 2. Average affinity deviation values.

Utilization bound	Average affinity deviation for BAF	Average affinity deviation for BFDC
1.6	0.039	9.727
2	0.144	12.518
2.4	0.473	14.484
2.8	1.011	16.613
3.2	1.926	19.064
3.6	2.992	22.261
4	4.130	25.698

7. FENP_MC: Fixed Execution Non-Preemptive Mixed Criticality

This section introduces a non-preemptive algorithm for MCSs, which executes in a time-triggered environment.

The task model used in this thesis is based on the periodical model for real-time tasks introduced in [21]. Here, tasks are called FModXs (Fixed Execution Executable Modules). Following the approach, a perfectly periodical task model is proposed for real-time systems:

$$M_i = \left\{T_i, D_i, L_i, \{C_{i,L_j} | \, j \in 1 \dots l\}, \{S_{i,L_j} | \, j \in 1 \dots l\}\right\} \tag{7-1}$$

where:

- M_i is a mixed criticality fixed execution task MC-FModX (Mixed Criticality Fixed Execution Executable Module);
- l represents the number of criticality levels;
- T_i is the period of periodical task i;
- D_i indicates the time by which any job execution needs to complete, relatively to its release time;
- L_i represents the criticality level;
- C_{i,L_j} is the worst-case execution time, WCET (vector of values – one per criticality level, for levels less than or equal to the criticality level L_i, expressing the worst-case execution time for each criticality level);
- S_{i,L_j} indicates the start time (vector of values one per criticality level, for levels less than or equal to the criticality level L_i, giving the execution start time, relative to its release time).

A task consists of a series of jobs, with each job inheriting the set of parameters of the task, (T_i, L_i, D_i), to which it adds its own parameters [22]. Thus, the k-th job of task M_i is characterized as:

$$J_{i,k} = \{a_{i,k}, d_{i,k}, c_{i,k}, s_{i,k}, T_i, D_i, L_i\} \tag{7-2}$$

where:

- $a_{i,k}$ is the arrival time ($a_{i,k+1} - a_{i,k} \geq T_i$);
- $d_{i,k}$ indicates the absolute deadline ($d_{i,k} = a_{i,k} + D_i$);

- $c_{i,k}$ represents the execution time allocated by the system, which is dependent on the criticality mode of the system (for L_j, $c_{i,k} = C_{i,L_j}$);
- $s_{i,k}$ gives the absolute execution start time for job k of task i, which is also dependent on the criticality mode of the system;
- T_i, D_i, L_i have the same meaning as in (7-1).

Definition 2. *The execution of task i is perfectly periodical if for each job k of task i, $J_{i,k}$, the difference between the absolute start times of jobs k and $k-1$ is constant:*

$$s_{i,1} - s_{i,0} = s_{i,2} - s_{i,1} = \cdots = s_{i,n} - s_{i,n-1} = T_i \tag{7-3}$$

Next, an exact feasibility test for the perfectly periodical execution of tasks in a non-preemptive context is presented. This type of execution is called Fixed Execution Non-Preemptive (FENP). The test is analogous with that provided in [21].

Let $M = \{M_1, M_2, \ldots, M_n\}$ be a set of n independent MC fixed execution tasks (MC-FModXs), sorted in nondecreasing order of their periods. The tasks are characterized by the same parameters as those in (7-1), thus:

$$M_i \equiv \left\{T_i, D_i, L_i, \{C_{i,L_j} \mid j \in 1 \ldots l\}, \{S_{i,L_j} \mid j \in 1 \ldots l\}\right\}, \quad \text{where for any task } k,\ T_k \leq T_i \text{ for } k < i \tag{7-4}$$

Definition 3. *The task set M is FENP schedulable in a mixed criticality system if, and only if, the task set M is FENP schedulable for each criticality level L_j, where $j \in 1 \ldots l$.*

Definition 4. *The task set M is FENP schedulable in a mixed criticality system for criticality level L_j if all the tasks in the set M with criticality greater than or equal to L_j are FENP schedulable using the next feasibility test. Only the parameters for level L_j (C_{i,L_j} and S_{i,L_j}) are considered in this case.*

The feasibility test for a task set executes as follows: tasks are sorted in nondecreasing order of their periods. Next, two tasks are extracted in order to compute the greatest common divisor. If the sum of the WCETs for the two tasks is greater than the greatest common divisor for a system execution mode, then the feasibility test is negative. If all the tasks in the set are successfully verified, then the feasibility test is positive.

Next, an adaptation to MCSs of the real-time table-driven scheduling algorithm FENP [21] for single processors and its partitioned P_FENP [23] variation for multicore systems is presented.

The FENP_MC scheduler creates, in an offline phase, a dispatch table for each criticality level of the system based on the feasibility test proposed in [21] for real-time systems, which was further developed and presented in this thesis for MCSs.

The dispatch table is represented by an array of structures:

$$\Gamma_q = \{TaskID;\ StartTime\} \tag{7-5}$$

where Γ_q is sorted in nondecreasing order of start time for each job in the system for a scheduling period.

8. P_FENP_MC: Partitioned Fixed Execution Non-Preemptive Mixed Criticality

P_FENP_MC consists of two phases, namely, an offline (Figure 4) phase and an online phase (Figure 5). The task partitioning to processors is carried out offline. A feasibility test is then conducted on each processor, followed by creating the scheduling table for that processor.

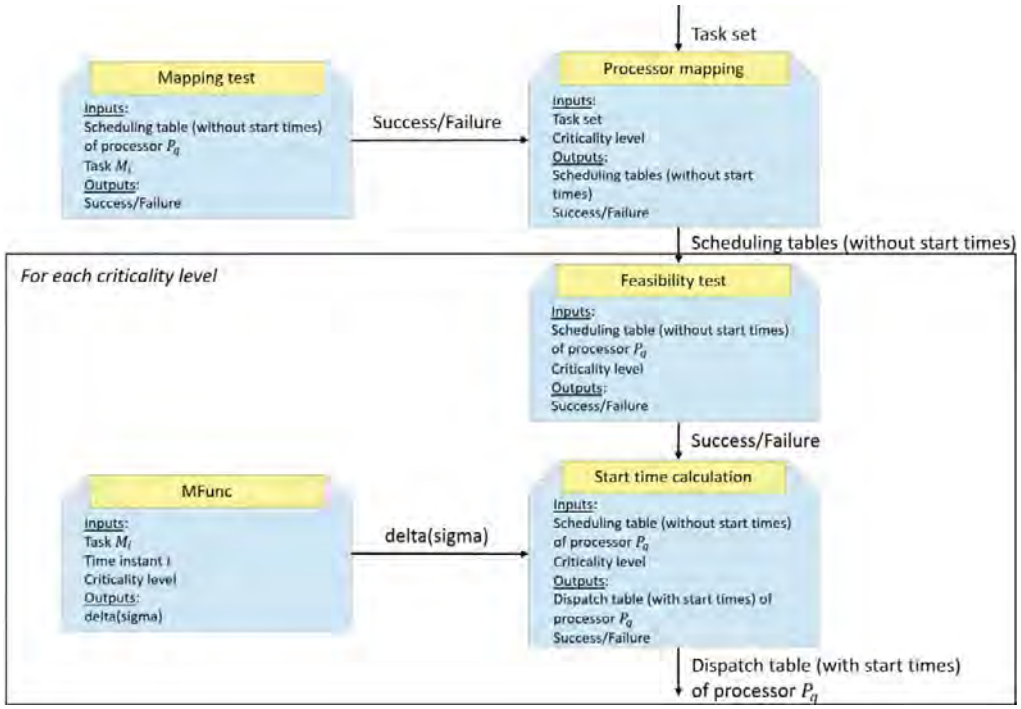

Figure 4. Offline phase execution.

The partitioning algorithm proceeds as follows:

Each processor has a scheduling table associated to it. Tasks from the ready queue are selected one by one and added in each scheduling table. If the scheduling table was initially not empty, two conditions are verified:

I. The current processor utilization, which is the sum of utilizations of all the tasks from the scheduling table associated with the corresponding processor, must not exceed 1 [24]:

$$U_{\Gamma_q} \leq 1, \text{ where } q = 1, \dots, m \tag{8-1}$$

II. The schedulability test performed for the task subset on the processor must be positive.

If the two conditions are met, the task will remain in the scheduling table, the processor utilization is updated, and the next task is removed from the ready queue and tested. If the scheduling table was initially empty, the task is added without verifying the two conditions and the processor utilization is updated.

If one of the two conditions returns FAILURE, the task is removed from the scheduling table and added in the next processor scheduling list, where the same test is performed.

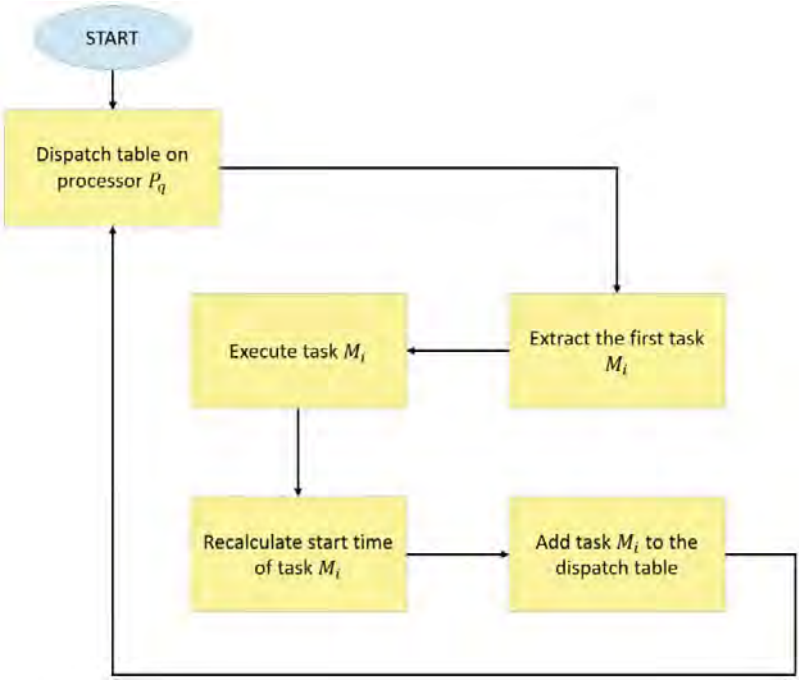

Figure 5. Online phase execution.

In the online phase tasks will be scheduled according to the dispatch tables. Initially, the system runs in Lo-criticality mode, therefore, tasks will be scheduled according to the Lo-criticality dispatch table. If a job exceeds its Lo-criticality WCET, the system will switch to Hi-criticality mode and tasks will be scheduled according to the Hi-criticality dispatch table. For each dispatch table tasks are ordered in a nondecreasing manner according to their start times. Next, the task with the lowest start time value M_i is extracted from the dispatch table, and its first job $J_{i,0}$ is executed. After job $J_{i,0}$ finishes executing, the start time of task M_i is recomputed. M_i will be added again in the corresponding dispatch table and the process is repeated (the task with the lowest start time value is extracted from the sorted task list and executed, and so on).

9. Performance evaluation

This chapter presents the experimental results obtained by evaluating the scheduling algorithm introduced in this thesis, namely Partitioned Fixed Execution Non-Preemptive Mixed Criticality (P_FENP_MC). The scheduling algorithm was compared in terms of success ratio with a known scheduling method for MCSs from the literature, P-EDF-VD (Partitioned Earliest Deadline First with Virtual Deadlines) [25], and with an adaptation for periodical tasks based on the table-driven algorithm introduced in [26], P-TT-OCBP (Partitioned Time-Triggered Own Criticality Based Priority). The task mapping to processors was done using FFD (First Fit Decreasing) [27], with sorting by period for both scheduling methods. Additionally, P_FENP_MC was compared in terms of jitter value with two time-triggered methods, TT-Merge (Time-triggered Merge) and Energy-efficient TT-Merge (Energy-efficient Time-triggered Merge) [28], and with the event-driven and table driven techniques described previously. The simulations were conducted on a homogenous system with two criticality levels, in a non-preemptive setting, using the simulation environment presented in Chapter 5.

9.2 Success ratio

As the number of processors increases, tasks are better scheduled in terms of success ratio when using the proposed algorithm, P_FENP_MC. With more available resources there is

a higher chance each task is partitioned on a suitable processor with regard to conditions I and II (see Chapter 8). The FFD does not run a schedulability test when mapping each task; therefore, if a high number of tasks are partitioned on a single processor, the local scheduling algorithm may return a negative schedulability test.

The tradeoff for jitterless scheduling on a uniprocessor is a lower success ratio value compared to using an event-driven method. An algorithm such as EDF-VD can reach up to 75% success ratio for a total utilization factor of 1 for the lowest criticality mode [29]. However, comparative results are harder to obtain with a time-triggered algorithm without using any resource enhancements, such as frequency scaling, for instance [26, 28].

For a multiprocessor platform, the success ratio is not only influenced by the scheduling algorithm but also by the function used to map tasks to processors. If a proper partitioned mapping function is used, comparative results can be obtained between time-triggered and event-driven schedulers in terms of success ratio.

9.3 Jitterless execution – Test Case

The jitter of a task is calculated as the difference between the maximum and minimum separation between two consecutive jobs of the same task τ_i [30]:

$$Jitter(\tau_i) = \max_{k\geq 1}\{|\, s_{i,k} - s_{i,k+1}|\} - \min_{k\geq 1}\{|\, s_{i,k} - s_{i,k+1}|\} \quad (9\text{-}1)$$

where:

- $s_{i,k}$ is the start time of job k of task τ_i.

Table 3 contains the jitter values obtained by applying expression (9-1) on a task set example with three tasks, scheduled using: P_FENP_MC, P-EDF-VD (a non-preemptive variant), TT-Merge, Energy-efficient TT-Merge and P-TT-OCBP.

Table 3. Jitter values of a task set example with three tasks, scheduled using five algorithms: P_FENP_MC, P-EDF-VD, TT-Merge, Energy-efficient TT-Merge and P-TT-OCBP.

Criticality mode	Task	Jitter value				
		P_FENP_MC	P-EDF-VD	TT-Merge	Energy-efficient TT-Merge	P-TT-OCBP
Lo	M_1	0	0	5	0	0
	M_2	0	1	1	1	1
	M_3	0	4	0	2	4
Hi	M_1	0	0	5	0	0

From the table above, it can be seen that four of the algorithms (P_FENP_MC, P-EDF-VD, Energy-efficient TT-Merge and P-TT-OCBP) provide jitterless execution of the first task, but only P_FENP_MC can deliver a jitterless execution of all the tasks in the system.

10. Conclusions and future work

The main contributions of this thesis are:

- A set of shared attributes of CPSs were identified and used for evaluating the scheduling algorithms for MCSs from the literature.
- A mixed criticality architecture was proposed for IoT (MC-IoT).
- A task execution model was defined for distributed real-time mixed criticality systems and a mapping methodology was introduced.

- A perfectly periodical scheduling algorithm for real-time mixed criticality systems was implemented.
- The simulation environment introduced in [15] for homogenous mixed criticality systems was modified in order to be used for heterogenous distributed mixed criticality systems.

As future perspectives:

- The scheduling algorithm can be implemented on a physical platform using LITMUS-RT, based on the task execution model introduced in this thesis.
- The algorithm can also be extended to systems with more than two criticality levels.
- Low criticality tasks can execute even after switching to a higher criticality level.
- A method can be implemented which enables the system to switch back to its initial execution mode, if some conditions are met.

Additionally, a series of improvements can extend the applicability of the task model and the scheduling algorithm to domains such as:

- Cyber physical systems – they use, for the most part, distributed mixed criticality platforms. Thus, the scheduling algorithm introduced in this thesis can be improved to allow adaptability and efficient resource management between different criticality levels. Furthermore, the proposed task model offers an efficient partitioning of applications at the device level in distributed platforms.
- Multi-agent systems – due to the hierarchical implementation, multiple classes of algorithms can be used, according to the requirements of each component. For the scheduling method introduced in this thesis, requirements can include: a perfectly periodical execution of certain tasks, efficient power management, synchronization between tasks, etc.
- IoT – the algorithm can be implemented in industrial control systems where certain critical applications have to be completely deterministic.

Bibliography

[1] R. Ernst and M. Di Natale, "Mixed criticality systems—A history of misconceptions?," *IEEE Design & Test,* vol. 33, no. 5, pp. 65-74, 2016.

[2] D. Tămaş–Selicean, P. Pop, and W. Steiner, "Design optimization of TTEthernet-based distributed real-time systems," *Real-time systems,* vol. 51, no. 1, pp. 1-35, 2015.

[3] J. Zhan, X. Zhang, W. Jiang, Y. Ma, and K. Jiang, "Energy optimization of security-sensitive mixed-criticality applications for distributed real-time systems," *Journal of Parallel and Distributed Computing,* vol. 117, no. pp. 115-126, 2018.

[4] P. A. Laplante, *Real-time systems design and analysis* vol. 3: Wiley New York, 2004, ISBN:

[5] K. Velasquez, D. P. Abreu, M. R. Assis, C. Senna, D. F. Aranha, L. F. Bittencourt, N. Laranjeiro, M. Curado, M. Vieira, and E. Monteiro, "Fog orchestration for the internet of everything: state-of-the-art and research challenges," *Journal of Internet Services and Applications,* vol. 9, no. 1, p. 14, 2018.

[6] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, 2007, pp. 239-243.

[7] A. Crespo, A. Alonso, M. Marcos, A. Juan, and P. Balbastre, "Mixed criticality in control systems," *IFAC Proceedings Volumes,* vol. 47, no. 3, pp. 12261-12271, 2014.

[8] M. A. Awan, K. Bletsas, P. F. Souto, and E. Tovar, "Semi-partitioned mixed-criticality scheduling," in *International Conference on Architecture of Computing Systems*, 2017, pp. 205-218.

[9] A. Ali and K. H. Kim, "Cluster-based multicore real-time mixed-criticality scheduling," *Journal of Systems Architecture,* vol. 79, no. pp. 45-58, 2017.

[10] P. Rodriguez, L. George, Y. Abdeddaïm, and J. Goossens, "Multicriteria evaluation of partitioned edf-vd for mixed-criticality systems upon identical processors," in *Workshop on Mixed Criticality Systems*, 2013.

[11] L. Gu, D. Zeng, S. Guo, A. Barnawi, and Y. Xiang, "Cost efficient resource management in fog computing supported medical cyber-physical system," *IEEE Transactions on Emerging Topics in Computing,* vol. 5, no. 1, pp. 108-119, 2015.

[12] D. B. Rawat, C. Brecher, H. Song, and S. Jeschke, *Industrial Internet of Things: Cybermanufacturing*

Systems: Springer, 2017, ISBN: 3319425587.
[13] R. Squire and H. Song, "Cyber-physical systems opportunities in the chemical industry: A security and emergency management example," *Process Safety Progress,* vol. 33, no. 4, pp. 329-332, 2014.
[14] N. Mohan and J. Kangasharju, "Edge-Fog cloud: A distributed cloud for Internet of Things computations," in *2016 Cloudification of the Internet of Things (CIoT)*, 2016, pp. 1-6.
[15] A. Sabu, B. Raveendran, and R. Ghosh, "SMILEY: a mixed-criticality real-time task scheduler for multicore systems," in *2018 IEEE/ACM 22nd International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, 2018, pp. 1-5.
[16] H. Li and S. Baruah, "Outstanding paper award: Global mixed-criticality scheduling on multiprocessors," in *2012 24th Euromicro Conference on Real-Time Systems*, 2012, pp. 166-175.
[17] N. Guan, P. Ekberg, M. Stigge, and W. Yi, "Improving the scheduling of certifiable mixed-criticality sporadic task systems," *Technical Report 2013–008,* no. 2013.
[18] I. Lupu, P. Courbin, L. George, and J. Goossens, "Multi-criteria evaluation of partitioning schemes for real-time systems," in *2010 IEEE 15th Conference on Emerging Technologies & Factory Automation (ETFA 2010)*, 2010, pp. 1-8.
[19] A. Alahmadi, A. Alnowiser, M. M. Zhu, D. Che, and P. Ghodous, "Enhanced first-fit decreasing algorithm for energy-aware job scheduling in cloud," in *2014 International Conference on Computational Science and Computational Intelligence*, 2014, pp. 69-74.
[20] Z. Ren, T. Lu, X. Wang, W. Guo, G. Liu, and S. Chang, "Resource scheduling for delay-sensitive application in three-layer fog-to-cloud architecture," no.
[21] M. V. Micea, V.-I. Cretu, and V. Groza, "Maximum predictability in signal interactions with HARETICK kernel," *IEEE transactions on instrumentation and measurement,* vol. 55, no. 4, pp. 1317-1330, 2006.
[22] L. Zeng, C. Xu, and R. Li, "Partition and Scheduling of the Mixed-Criticality Tasks based on Probability," *IEEE Access,* vol. 7, no. pp. 87837-87848, 2019.
[23] E. A. Capota, C. S. Stangaciu, M. V. Micea, and V. I. Cretu, "P_FENP: A Multiprocessor Real-Time Scheduling Algorithm," in *2018 IEEE 12th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, 2018, pp. 000509-000514.
[24] D. Socci, "Scheduling of certifiable mixed-criticality systems," Grenoble Alpes, 2016.
[25] J.-J. Han, X. Tao, D. Zhu, H. Aydin, Z. Shao, and L. T. Yang, "Multicore mixed-criticality systems: Partitioned scheduling and utilization bound," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 37, no. 1, pp. 21-34, 2017.
[26] S. Baruah and G. Fohler, "Certification-cognizant time-triggered scheduling of mixed-criticality systems," in *2011 IEEE 32nd Real-Time Systems Symposium*, 2011, pp. 3-12.
[27] B. Rieck, "Basic analysis of bin-packing heuristics," *Publicado por Interdisciplinary Center for Scientific Computing. Heildelberg University,* no. 2010.
[28] L. Behera and P. Bhaduri, "An energy-efficient time-triggered scheduling algorithm for mixed-criticality systems," *Design Automation for Embedded Systems,* no. pp. 1-31, 2019.
[29] S. Baruah, V. Bonifaci, G. DAngelo, H. Li, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie, "The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems," in *2012 24th Euromicro Conference on Real-Time Systems*, 2012, pp. 145-154.
[30] S. Baruah, G. Buttazzo, S. Gorinsky, and G. Lipari, "Scheduling periodic task systems to minimize output jitter," in *Proceedings Sixth International Conference on Real-Time Computing Systems and Applications. RTCSA'99 (Cat. No. PR00306)*, 1999, pp. 62-69.