Article

A Model-Based Approach to Optimizing Partition Scheduling of Integrated Modular Avionics Systems

Pujie Han ^{1,*}, Zhengjun Zhai ¹ and Lei Zhang ²

¹ School of Computer Science and Engineering, Northwestern Polytechnical University, Xi'an 710072, China; zhaizjun@nwpu.edu.cn

² Xi'an Aeronautics Computing Technique Research Institute, AVIC, Xi'an 710068, China; zlei3@avic.com

* Correspondence: hanpujie@mail.nwpu.edu.cn

Received: 9 July 2020; Accepted: 6 August 2020; Published: 9 August 2020

Abstract: The architecture of Integrated Modular Avionics (IMA) provides airborne software with a robust temporal partitioning mechanism, which achieves the reliable fault containment between avionics applications. However, the partition scheduling of an IMA system is a complex nonlinear non-convex optimization problem, making it difficult to solve the optimal temporal allocation for partitions using traditional analytical methods. This paper presents a model-based approach to optimizing the partition scheduling of IMA systems, whose temporal behavior is modeled as a network of timed automata. Given a system model, the optimizer employs a parallel genetic algorithm to search for the optimal partition resource parameters with respect to minimum processor occupancy. For each promising parameter combination, the schedulability constraints and processor occupancy of the system are precisely evaluated by Classical and Statistical Model Checking (i.e., CMC and SMC), respectively. We also apply SMC hypothesis testing to the fast falsification of non-schedulable solutions, thereby speeding up the schedulability verification based on CMC. Two case studies demonstrate that our proposed approach outperforms classical analytical methods on the processor occupancy of typical IMA systems.

Keywords: model-based optimization; integrated modular avionics; partition scheduling; timed automata; parallel genetic algorithm; schedulability verification; model checking

1. Introduction

As the microprocessor technology advances considerably over recent years, there is a growing trend towards integrating more airborne real-time applications into a generalized COTS (commercial off-the-shelf) computer in the avionics industry [1]. For this purpose, the architecture of integrated modular avionics (IMA) is widely applied to the design of advanced aircraft. An IMA system comprises a number of standardized computing modules that are capable of supporting various airborne applications, each of which accommodates a set of concurrent tasks and runs on a partitioned operating system (OS), as prescribed by the ARINC 653 standard [2]. The ARINC-653 OS provides each application with a strict temporal partition that is scheduled independently of its internal tasks, thereby achieving reliable fault containment between applications. As a result, this temporal partitioning mechanism poses a challenge of how to schedule the partitions in an optimal manner while guaranteeing their schedulability constraints.

The temporal partitioning features of IMA are commonly characterized as a two-level hierarchical scheduling system, where partitions are scheduled by a time division multiplexing (TDM) global scheduler on the basis of a periodic resource model, and the local scheduler of a partition manages its internal tasks in accordance with their priorities [3]. From this perspective, the partition scheduling problem of an IMA system is to find the optimal parameters of partition resource model that

minimizes processor usage while guaranteeing the schedulability at both the global and local level. However, the optimization of multiple partitions in hierarchical scheduling is a complex nonlinear non-convex optimization problem [4], making it difficult to solve the optimal temporal allocation for partitions by using traditional analytical methods.

Related analytical methods attempt to simplify this non-convex optimization problem, owing to the weak expressiveness of classical analytical models. The authors of [5–8] adopt compositional policies that calculate the schedulability bound for each partition independently on the worst-case assumption of the rest of the partitions, combining all of the local optima as a complete partition resource model of the system. Davis et al. [9] investigate the holistic design of multiple periodic partitions in hierarchical fixed-priority scheduling systems, deriving schedulability conditions from a response time analysis (RTA), performing a heuristic search to find optimal schedulable resource parameters. Yoon et al. [4] solves the global optimization problem via geometric programming (GP), where the schedulability constraints are defined as the inequality relations between approximate demand and supply bound function. Blikstad et al. [10] simplified the hierarchical scheduling mechanism into non-stratified TDM scheduling of non-overlapping tasks, employing mixed integer linear programming (MILP) to produce the optimal static scheduling table. Nevertheless, the analytical models, such as RTA and GP, introduce oversimplification into the schedulability formulation of the system, leading to unnecessary waste of processor time.

By contrast, formal models are more expressive to describe the complex temporal behavior of IMA systems [11]. Various model-based methods build on different formalism, such as preemptive time Petri nets (pTPN) [3,12], satisfiability modulo theories (SMT) [13], linear hybrid automata (LHA) [14], and timed automata (TA) [15–18]. They mainly focus on the schedulability verification of IMA systems whose partition scheduling configuration is predetermined or tentatively selected, but none of the studies address the global optimization problem of IMA partition scheduling, because it is inefficient to explore the entire combinatorial solution space that is associated with the formal models. As a growing number of computing-intensive aircraft functions are being integrated into heavily loaded IMA systems, the model-based method is increasingly required to cope with the high-dimensional solution space with higher precision, providing a tight processor utilization bound for schedulability.

In this paper, we propose a model-based approach for optimizing the partition scheduling of IMA systems, searching for the optimal partition resource parameters with respect to the minimum processor occupancy. The proposed approach describes the temporal behavior of IMA systems as a set of TA models that is more expressive and precise than traditional analytical models, thus avoiding the oversimplification of this non-convex optimization problem. When compared with related model-based work, our approach combines the parallel genetic algorithm and simulation-based testing to perform a more efficient exploration of the high-dimensional solution space of heavily loaded IMA systems with high precision. Our main contributions include:

- A model-based optimization method that solves the optimal partition scheduling of IMA systems by carrying out an efficient search of the global solution space of the TA system model via parallel genetic algorithm.
- The TA modeling of IMA partition scheduling systems that describes the temporal partitioning behavior of an IMA system as a hierarchical scheduling model, which comprises a set of communicating TA templates.
- A fast parallel schedulability analysis, where each promising solution undergoes an independent two-step schedulability analysis that includes fast falsification based on statistical model checking (SMC) and verification based on classical model checking (CMC).
- An SMC-based evaluation of processor occupancy that employs the SMC simulation to calculate the expectation of partition context-switching overhead, thereby evaluating the processor occupancy precisely.

This paper is organized, as follows. The system model and optimization problem are first defined in Section 2. Section 3 introduces the design of our model-based optimization method. Section 4

describes the TA models of IMA partition scheduling systems. In Section 5, we present the application of parallel genetic algorithm, the schedulability analysis method, and the SMC-based evaluation of processor occupancy. In Section 6, two experiments are provided in order to demonstrate that our model-based approach outperforms the related analytical optimization methods. Conclusions are finally drawn in Section 7.

2. Definition of Optimization Problem

This section provides the optimization problem definitions of IMA partition scheduling, whose system model is defined according to the ARINC 653 standard [2]. On the basis of this system model, the optimization problem of IMA partition scheduling is formulated as the parameter optimization of periodic partitions in shierarchical scheduling architecture.

2.1. System Model

We consider a typical IMA system that is equipped with two-level schedulers and corresponding execution entities: the system is divided into a set of temporal partitions, which are scheduled by a TDM global scheduler according to a static partition schedule. The tasks that belong to a partition are scheduled by a preemptive fixed priority (FP) local scheduler [3]. Both the partitions and tasks are viewed as execution entities. A task is allowed to run only when its partition is activated by the global scheduler. Figure 1 presents an example of such an IMA partition scheduling system, where the TDM global scheduler and FP local schedulers manage two partitions and five tasks, respectively.

Figure 1. An example of partition scheduling systems of integrated modular avionics (IMA).

Let $\Omega = \{P_i | i = 1, 2, \dots, n\}$ be a set of partitions running on a single processor. The global scheduler assigns processor time to partitions in accordance with the static schedule $\mathcal{S}$ cyclically and repeats the allocation of $\mathcal{S}$ every major time frame M [2]. The partition schedule $\mathcal{S}$ consists of a set of partition time windows: $\mathcal{S} = \{W_t | t = 1, 2, \dots, w\}$, where W_t is a time slot $\langle P_t, o_t, d_t \rangle$ of a partition $P_t \in \Omega$. o_t and d_t represents the offset from the start of M and execution duration, respectively. The w time slots must be non-overlapping, satisfying that $0 \leq o_1 < o_1 + d_1 < o_2 < o_2 + d_2 < \dots < o_w < o_w + d_w \leq M$.

Let $\Gamma_i = \{\tau_j^i | j = 1, 2, \dots, m_i\}$ be the task set of a partition P_i . A task τ_j^i is defined as the quadruple $\langle T_j^i, E_j^i, D_j^i, R_j^i \rangle$ where T_j^i is task period, E_j^i is the worst-case execution time of a job of the task, $D_j^i \leq T_j^i$ denotes (relative) deadline, and R_j^i denotes task priority. The task τ_j^i is initially released at the start of P_i 's first partition time window. We refer to this instant as the initial offset I_j^i of τ_j^i . Hence, τ_j^i releases the k th job at the instant $t_k = I_j^i + kT_j^i, k \in \mathbb{N}$. The task τ_j^i meets its deadline if and only if τ_j^i can finish its k th job before the instant $(I_j^i + kT_j^i + D_j^i)$ for any $k \in \mathbb{N}$. The schedulability of this real-time system requires all tasks to meet the deadlines.

The ARINC 653 standard defines a partition resource model $\langle p_i, b_i \rangle$ that denotes the periodic processor-time demand of a partition $\mathcal{P}_i$, where p_i is a partition period, b_i is the budget within p_i , and the budget b_i should be guaranteed by the partition schedule $\mathcal{S}$ during each period p_i . This parameter model describes each partition as a periodic execution unit, thus providing an interface between the system integrator and software function suppliers [6,19].

Given a specific scheduling policy, the parameters of partition resource models determine the allocation of partition time windows [2]. In this paper, we also adopt the preemptive FP scheduling policy for partitions, whose priorities are preassigned according to their critical levels.

2.2. Optimization Problem

Consider a two-level hierarchical scheduling system that adopts the preemptive FP policy in order to schedule both the set of partitions $\Omega = \{\mathcal{P}_i | i = 1, 2, \dots, n\}$ and their task sets $\Gamma_i = \{\tau_j^i | j = 1, 2, \dots, m_i\}$.

The optimization problem of partition scheduling is to find a $2n$ -dimensional vector $\mathbf{x} = (x_1, x_2, \dots, x_{2n}) \in \mathbb{R}_+^{2n}$ where the partition resource model $\langle p_i, b_i \rangle$ of $\mathcal{P}_i$ corresponds to the elements $x_{2i-1} = p_i$ and $x_{2i} = b_i$, such that the system minimizes the processor occupancy U while satisfying the schedulability constraints that all of the tasks meet their deadlines.

Assume that each release of partitions involves a context switch. The definition of processor occupancy is

$$U = \sum_{i=1}^n \frac{\bar{c}_i \times v + b_i}{p_i}, \quad (1)$$

where $\bar{c}_i$ is the average number of context switches for $\mathcal{P}_i$ during each partition period p_i , and v is the overhead of a context switch.

The IMA partition scheduling system has an equivalent definition of processor occupancy on the basis of the TDM global scheduling policy. We deduce this equivalent relation, as follows.

According to the ARINC 653 standard, the major time frame M is the least common multiple of all the partition periods p_i [2]. Without loss of generality, assume that $M = k_i p_i, k_i \in \mathbb{N}_+$. The TDM scheduling policy of an IMA system satisfies the following relation:

$$\sum_{t=1}^w d_t = \sum_{i=1}^n \sum_{j=1}^{k_i} b_i, \quad (2)$$

which represents the balance between temporal supply (the schedule $\mathcal{S}$) and demand (the unknown parameter $\langle p_i, b_i \rangle$).

Additionally, the number of partition context switching of $\mathcal{S}$ is equal to that of partition time windows within the major time frame M :

$$w = \sum_{i=1}^n \sum_{j=1}^{k_i} c_j^{(i)}, \quad (3)$$

where $c_j^{(i)}$ is the number of context switching of the partition $\mathcal{P}_i$ during the j th period of M .

Thus, we can define the objective function $U(\mathbf{x})$ on the basis of the partition schedule $\mathcal{S}$:

$$\begin{aligned}
 U &= \sum_{i=1}^n \frac{b_i + \bar{c}_i \times v}{p_i} \\
 &= \sum_{i=1}^n \frac{1}{p_i} \left(b_i + \sum_{j=1}^{k_i} \frac{c_j^{(i)}}{k_i} \times v \right) \\
 &= \sum_{i=1}^n \frac{1}{k_i p_i} \left(k_i b_i + \sum_{j=1}^{k_i} c_j^{(i)} \times v \right) \\
 &= \sum_{i=1}^n \sum_{j=1}^{k_i} \frac{b_i + c_j^{(i)} \times v}{M} \\
 &= \sum_{t=1}^w \frac{d_t + v}{M}, \tag{4}
 \end{aligned}$$

which is the processor occupancy of all the partition time windows and their context switching within a major time frame M of the schedule $\mathcal{S}$. In this case, the same task sets with specific time requirement $\langle p_i, b_i \rangle$ always produce the same processor occupancy of partition schedules.

Therefore, the IMA partition scheduling optimization problem is to find the optimal parameter vector of $\langle p_i, b_i \rangle$, which provides the minimum processor occupancy in Equation (1) while guaranteeing the schedulability constraints.

3. Model-Based Optimization Method

The core idea of our model-based optimization method is to run a heuristic search of the partition resource parameters of the TA system model. Because it is difficult to solve the derivative information of such a complex formal model, a heuristic search algorithm is used here to explore the parameter space.

Figure 2 shows the flowchart of the model-based optimization method. The dash-line rectangle on the left depicts the data flow of the optimization process, which consists of the following five steps:

1. The search algorithm explores the parameter space of partition resource models according to the definition domain and the knowledge acquired during the search process, finally generating a set of promising parameter vectors $\mathcal{G}$.
2. For each valid parameter vector $\mathbf{x}$, we create the TA model $\mathcal{M}$ that declares $\mathbf{x}$ as arrays of integer variables. The grammar of these TA models is subject to the regulation of the model checker UPPAAL [20].
3. Each of the TA models undergoes an SMC hypothesis test in order to fast falsify the non-schedulable vectors of $\mathcal{G}$. The SMC tests are conducted in UPPAAL SMC [21], which executes numerous simulation processes, offering the statistical results of schedulability with a high degree of confidence. All of the parameter vectors that get through the SMC test are regarded as statistically schedulable.
4. Subsequently, UPPAAL SMC is invoked to evaluate the processor occupancy of these statistically schedulable models. The heuristic search will finish and append the best parameter vectors to a candidate set $\mathcal{C}$ if the termination criterion is reached. Otherwise, the optimizer will return to Step 1 that proceeds with the exploration.
5. After finishing the heuristic search, our optimizer verifies the schedulability of all the candidate vectors in $\mathcal{C}$ by model checking in UPPAAL Classic. If any satisfactory solutions are found, the optimization will be completed and output the optimal solution $\mathbf{x}_{opt}$ in $\mathcal{C}$. If not, it will repeat the above exploration-and-evaluation steps until an optimal solution is reached.

Figure 2. Flowchart of model-based optimization method.

The TA system model provides more precise schedulability bounds than traditional analytical methods, owing to the strong expressiveness of TA. We apply a parallel genetic algorithm to the heuristic search in the model-based optimization. The integration of fast SMC tests and parallel genetic algorithm improves the efficiency of parameter space exploration.

In the following sections, we present two major constituent components of this model-based method: the TA model and the parallel genetic algorithm with a fast schedulability analysis.

4. TA Modeling of IMA Partition Scheduling Systems

This section presents a TA modeling framework of IMA partition scheduling systems. The TA model is created in UPPAAL, which provides a modeling environment for the TA extended with data types, such as bounded integers, arrays, etc. [22]. UPPAAL defines the TA model of a real-time system as a set of Templates, each of which is comprised of locations and edges and can be instantiated as a series of TA instances. Different TA instances communicate with each other via the channels on edges. The communicating TA constitute a complete system model. In this section, we first introduce the modeling framework and then detail the TA templates of the system model.

4.1. TA Modeling Framework

The TA model of an IMA partition scheduling system is organized as a stratified structure that consists of four layers: the TA instances of the following four templates *GlobalScheduler*, *Partition*, *LocalScheduler*, and *Task*. Figure 3 shows the layered templates and the channels between them.

These channels are used as the scheduling commands of schedulers (global/local scheduler) and the requests of execution entities (partitions/tasks).

Figure 3. Timed automata (TA) modeling framework of IMA partition scheduling systems.

When considering the IMA system in Section 2.1, we create one TA instance of the template GlobalScheduler, n Partition, n LocalScheduler, and $\sum_{i=1}^n m_i$ Task instances. The number of partitions and tasks are defined as two integer types `pid_t` and `tid_t`, respectively.

The template GlobalScheduler realizes the function of a global preemptive FP scheduler that manages all the partitions. The global scheduler model communicates with partition models via four channels: A partition receives the input signal `psched` or `pstop` when it is scheduled to run or preempted by other partitions, respectively. The global scheduler is requested to execute a partition through `pready`. Finally, the activated partition uses `prelease` in order to relinquish the processor.

The template Partition describes the current state of a partition that is viewed as an execution unit scheduled by the global scheduler. Once the system enters or leaves a partition, the local scheduler should receive an input signal `enter_partition` or `exit_partition`, respectively.

The template LocalScheduler is a generic model of the local scheduler of a partition to control the execution of tasks inside the partition. The channels between LocalScheduler and Task are similar to those between GlobalScheduler and Partition. The signal `sched`, `stop`, `ready`, and `release` mean a task is scheduled, preempted, getting ready to run, and relinquishing the processor, respectively.

The template Task acts as the basic execution entities in the system, i.e., the real-time tasks of each partition. Tasks are managed by the local schedulers of their partitions via the channels `sched` and `stop`. In the opposite direction, a Task model notifies its state to the local scheduler through `ready` and `release`.

The stratified structure provides an extensible modeling design. It is convenient to replace any template of the modeling framework with a new one that only needs to conform with the same definition of the inter-layer channels as that in this paper. For example, a different local scheduling policy can be implemented by a new LocalScheduler template in order to replace the original preemptive FP policy.

4.2. Global Scheduler Template

The global scheduler template GlobalScheduler adopts a preemptive fixed priority scheduling algorithm to allocate processor time to all partitions. It declares a partition ready queue `rq` and inserts the identifiers of all the partitions ready to run into this queue in order of priority. The partition ready queue `rq` is associated with a set of operation functions: the function `enqueue` and `dequeue` insert a partition

identifier and remove the head element, respectively; the function `front` fetches the first partition identifier from `rq`. The length of `rq` is returned by the function `pqLen`.

Figure 4 depicts the global scheduler template, whose execution starts from the initial location **Idle**. When receiving a signal of **pready** from a ready partition `n`, the model calls `enqueue` to insert the identifier `n` into `rq` and migrates to the committed location **Schedule**. Because UPPAAL freezes time at committed locations (i.e., the locations internal marked “C”), the scheduler will leave the **Schedule** location immediately, subsequently scheduling the first partition of `rq` to run through the **psched** channel and moving to the location **Occupied**.

Figure 4. Global scheduler template.

At the **Occupied** location, the global scheduler model is able to handle the possible preemption. Once a **pready** input is received from the recently ready partition, the model updates the ready queue `rq` and moves to the location **Preempt**, where the scheduler checks whether the head element of `rq` is still the running partition. If so, the model will only return to **Occupied**. Otherwise, a preemption operation will be performed: the running partition is stopped by the **pstop** channel. The model then enters the **Schedule** location to execute the new head element of `rq`.

When a partition finishes its execution, it sends the global scheduler model a **prelease** notification in order to release the processor. The model will migrate from **Occupied** to the location **Release** and remove the current partition from the ready queue `rq`. If `rq` is empty at this instant, the model will return to **Idle** and wait for a new **pready** request. Otherwise, it will enter **Schedule** to run the other partitions of `rq`.

4.3. Partition Template

In the optimization problem, each partition $\mathcal{P}_i$ is viewed as an execution entity and is further described as the partition resource model $\langle p_i, b_i \rangle, i = 1, 2, \dots, n$. In the TA system model, the parameter vector $\mathbf{x}$ of $\langle p_i, b_i \rangle$ is declared as two integer arrays `partprd` and `partbudg`, where p_i and b_i are recorded in `partprd[i]` and `partbudg[i]`, respectively. The Partition template defines its partition identifier i as a `pid_t` type variable `pid`. The overhead of each context switch is assigned to the constant `cs`. For each partition period, the number of context switches is recorded in the integer array `csCnt`, where the array index is the partition identifier `pid`.

There are three clock variables in the Partition template. The first clock `curTime` measures the time during a period. The second clock `exeTime` indicates the execution time in the current period. If the partition does not occupy the processor, `exeTime` will be suspended by assigning zero to the derivative of `exeTime` (i.e., `exeTime'`). The third clock `x` measures the time during a partition context switch.

Figure 5 shows the Partition template. All partitions are released at the initial location **Init**, sending a **pready** request to the global scheduler and then waiting for the scheduling command **psched** at the location **WaitPartition**. After receiving a **psched** signal with the partition identifier **pid**, the model undergoes a context switch at the location **Schedule** for the duration **cs**. The partition **pid** is entered after the context switch, moving to the location **InPartition**, and notifying its local scheduler via the channel **enter_partition**.

Figure 5. Partition template.

Partitions may be preempted at the location **InPartition**. Before the clock **exeTime** reaches the partition budget, receiving a **pstop** command transfers the model to the location **Preempt**, where the partition exits and relinquishes the processor. Meanwhile, the model sends a partition-leaving notification to the local scheduler via the channel **exit_partition**. If the budget is satisfied at **InPartition**, the partition will also exit through **exit_partition**, migrating from the location **Complete** to **WaitNextPeriod**. The model will stay at **WaitNextPeriod** until the beginning of the next period, when a new ready request will be produced to repeat the above scheduling process starting from **WaitPartition**.

In addition, the Partition template defines an error location **LackOfSupply**, which should be entered once the model violates the budget constraint within a period. The function **setError** on the incoming edges assigns *True* to a boolean variable **Error** that indicates the schedulability of the system. At **LackOfSupply**, receiving a scheduling command **psched** brings about the instant output on **prelease**, immediately releasing the processor and avoiding deadlock.

4.4. Local Scheduler Template

This paper uses the same **LocalScheduler** template that was introduced in our previous work [23] and subject to the definition of the adjacent channels presented in Figure 3.

The **LocalScheduler** template is an event-driven model. It always reacts to a particular set of input channels. The **LocalScheduler** models receive partition-entering and -exiting notification from the **GlobalScheduler** model via two channels **enter_partition** and **exit_partition** respectively. Additionally, four other channels **ready**, **release**, **sched**, and **stop** are used as scheduling requests or commands to manage the tasks of partitions.

4.5. Task Template

Figure 6 presents the Task template, which is associated with a task identifier *tid*. A Task model stays at the initial location **Init** until the task is first released at the instant when it receives the first **enter_partition** signal from its Partition model. After being released, the task starts its first period and gets ready to run at the location **Ready**, sending a **ready** request to its local scheduler.

Figure 6. Task template.

The task models are controlled by its local scheduler to shuttle between the location **Ready** and **Running**. When receiving the command **sched** or **stop** from the local scheduler, the task is scheduled to run or be preempted by other tasks, respectively.

The clock *exeTime* measures the execution time within a task period, thus stopping progressing at **Ready** by using the invariant “*exeTime*’ == 0”. The function *execTime* returns the expected duration of each periodic job. Once the expected execution is completed (i.e., the guard “*exeTime* ≥ *execTime*()” is satisfied), the model relinquishes the processor via the channel **release** and then enters the location **WaitNextRelease** to wait for the next period.

By contrast, the clock *curTime* never stops. It indicates the time during a whole period to determine whether the task meets its deadline in current period. The function *deadline* provides the relative deadline of the task. If the deadline is violated (i.e., the guard “*curTime* ≥ *deadline*()” is satisfied), the model will immediately move to the location **MissDeadline**, assigning *True* to the boolean variable *Error* in the function *setError*.

Even if a task model falls into the **MissDeadline** location, it will still react to the **sched** scheduling command of the local scheduler by releasing the processor via the channel **release**. This prevents the task and its local scheduler from reaching a deadlock.

5. Application of Parallel Genetic Algorithm

A parallel genetic algorithm implements an efficient heuristic search in the model-based optimization of IMA partition scheduling. In this section, we first present the procedure for this parallel genetic algorithm, introduce a fast schedulability analysis, and finally describe the SMC-based evaluation of processor occupancy. The schedulability and processor occupancy constitute the fitness function that interfaces this parallel genetic algorithm with the model-based optimizer.

5.1. Procedure for Parallel Genetic Algorithm

In the optimization process, the heuristic search algorithm determines the direction of exploration of the parameter space to generate better candidate solutions in each iteration. This heuristic search is realized by a parallel genetic algorithm, which simulates the collective learning process of natural evolution.

Because each parameter vector and its TA model are processed independently in the optimization, the most costly model-based analysis can be accelerated via parallelization. The parallel genetic algorithm splits a candidate solution set together with their TA models into multiple sets and distributes them across nodes in a cluster, thereby processing the model-based analyses in parallel. This allows for the optimization to be processed more efficiently than it would be in a conventional sequential search.

Figure 7 shows the procedure for the parallel genetic algorithm of the model-based optimizer. This evolutionary algorithm consists of the following eight steps:

1. Initialization: according to the user-defined value range of partition resource parameters, the initial population Φ_0 is randomly generated. Φ_0 contains X individuals. Each individual in a population is a candidate parameter vector $\mathbf{x} \in \langle p_i, b_i \rangle^n$.
2. Division: the j th generation of the population $\Phi_j, j \geq 0$ is divided equally into m sub-populations $\{\Phi_j^k | k = 1, \dots, m\}$, each of which will be processed by Step 3~5 in parallel.
3. Crossover: x' pairs of parental individuals are mated randomly in the sub-population Φ_j^k . A crossover operator is invoked to produce a set of new individuals $\Phi_j^{k'}$ by combining the information of their parents. All the sub-populations generate $X' = mx'$ children.
4. Mutation: we call a mutation operator that alters the parameters of each individual with a low probability, mapping $\Phi_j^{k'}$ to the offspring set Ψ_j^k .
5. Output: the m sub-populations Ψ_j^k are output independently to the optimizer processes, which will continue to execute step 2 of Section 3 on the basis of the parameter vector set $\mathcal{G}$ contained in the population $\Psi_j = \{\Psi_j^k | k = 1, \dots, m\}$.
6. Merging: we collect from the optimizer processes the results of schedulability analyses and processor occupancy evaluation of the population Ψ_j .
7. Fitness calculation: we calculate the fitness value of each individual in the population Ψ_j on the basis of its schedulability and processor occupancy.
8. Selection: a selection operator is invoked to select the best X individuals from the population Ψ_j according to the fitness values. These X offspring individuals constitute the next generation of the population Φ_{j+1} . After this selection operation, we enter the $(j + 1)$ th iteration by executing $j = j + 1$ and return to Step 2.

Figure 7. Procedure for the parallel genetic algorithm of the model-based optimizer.

We propose the following schedulability analysis and processor occupancy evaluation in order to provide an efficient fitness calculation for the parallel genetic algorithm.

5.2. Schedulability Analysis

As depicted in Figure 8, a combination of CMC and SMC is applied to the model-based schedulability analysis, which consists of two steps:

1. For each system model $\mathcal{M}$ encoding a parameter vector $\mathbf{x}$, a UPPAAL SMC hypothesis test is performed to determine whether the schedulability constraint is satisfied with high probability. UPPAAL SMC also evaluates the expected value of processor occupancy of $\mathcal{M}$.
2. Only the system models that get through the SMC test can be checked by the CMC verification in UPPAAL Classic to confirm their schedulability.

Figure 8. Schedulability analysis of the model-based optimizer.

Because SMC is based on the Monte-Carlo simulation, its processing is more efficient than the state-space exploration method of CMC [17]. The application of SMC not only avoids the exhaustive exploration of the state space of TA models, but also quickly finds the counter-examples that violate the schedulability constraints. Thus, most of the non-schedulable parameter vectors can be quickly excluded from the set of candidate solutions.

However, the nature of SMC is statistical simulation-sampling testing, making it impossible to prove the schedulability strictly. Therefore, CMC is still required to confirm the schedulability of few candidate solutions that pass SMC tests.

In order to facilitate the schedulability analysis, we define a boolean variable *Error* with an initial value *False* in the TA models. This *Error* is the non-schedulability indicator of a TA model: once any of the schedulability constraints is violated, the model should execute the operation *Error=True* on related edges immediately.

In UPPAAL SMC, the schedulability constraints of a system model are described as the equation of hypothesis testing:

$$\Pr[\leq M](\Box \text{ not Error}) \geq \theta, \quad (5)$$

which is the null hypothesis denoting that the probability of satisfying schedulability is greater than θ . M is the time limit of the simulation processes. θ is a relatively large probability value and normally $\theta \geq 0.99$.

Consider two types of errors that result from the hypothesis test:

- Type I error: a Type I error occurs when UPPAAL SMC rejects the null hypothesis of Equation (5) that is true. This leads to a false positive conclusion, mistakenly excluding a schedulable solution in advance. The probability of committing a Type I error is defined as a significance level α .
- Type II error: a Type II error is the non-rejection of the false null hypothesis of Equation (5). This brings about a false negative conclusion, possibly making the candidate set drowning in non-schedulable solutions. The probability of committing a Type II error is referred to as β .

The purpose of this hypothesis testing is to exclude as many non-schedulable vectors as possible, thereby reducing the processing time of the following costly CMC verification. In order to improve the efficiency of the model-based schedulability analysis, we use a smaller probability β of Type II error than the significance level α of hypothesis testing. Typically, $\alpha = 0.05$ and $\beta = 0.01$ are applied to the UPPAAL SMC hypothesis tests.

When we perform the CMC verification in UPPAAL Classic, the schedulability constraints are formulated as the safety property of timed computation tree logic (TCTL) [20]:

$$\forall \square \text{ not Error} \quad (6)$$

which means that Error is always False in the state space of a system model. Thus, the satisfaction of this safety property is able to guarantee the schedulability of the system.

5.3. Evaluation of Processor Occupancy

For each parameter vector $\mathbf{x} \in \langle p_i, b_i \rangle^n$, the evaluation of its processor occupancy requires calculating the number of context switches during each partition period. This can be achieved by the following equation of expectation evaluation in UPPAAL SMC:

$$E[<= M; N] \left(\max : \text{sum}(i : \text{pid_t}) \frac{(\text{csCnt}[i] \times \text{cs} + \text{partbudg}[i]) \times K}{\text{partprd}[i]} \right) \quad (7)$$

where M defines the time limit of simulation runs, N gives the number of runs, i is a partition identifier, and K is a constant factor that scales up the result of processor occupancy to be an integer. As defined in Section 4.3, the symbols `pid_t`, `csCnt`, `cs`, `partbudg`, and `partprd` are the type of partition identifiers, the number of context switches within a partition period, the overhead of a context switch, the partition budget, and the partition period, respectively.

Equation (7) evaluates the expected value U_e of the upper bound of processor occupancy within the first M time units. According to the results of the schedulability analysis and processor occupancy evaluation, we define the fitness function of the parallel genetic algorithm as

$$F = (C + 1) \times K - U_e \quad (8)$$

where C is an integer variable that will be 1 if the model of $\mathbf{x}$ is schedulable and 0 otherwise.

The fitness function $F : \mathbf{x} \rightarrow \mathbb{N}$ provides the quality evaluation of any parameter vector $\mathbf{x}$, thereby determining the search directions of the parallel genetic algorithm. A higher fitness value represents a better individual that is more likely to be selected in the evolution.

6. Case Studies

This section applies our model-based method to the optimization of two representative IMA partition systems, which were introduced in [9]. We adopt four typical operator combinations to run an optimization experiment on a cluster. We attempt to show the feasibility of our method as well as find a suitable operator combination for the optimization problem. A comparison experiment is also carried out in order to demonstrate the better performance of our model-based method than traditional analytical ones.

6.1. Implementation of Model-Based Method

The model-based optimizer is implemented as a Python program that is deployed and executed on a high-performance cluster. Table 1 shows the detailed configuration of this experimental platform.

Table 1. The configuration of the experimental platform.

Item	Configuration
Number of nodes	4
Number of processors	4/node
Processor type	AMD Opteron 6376, 2.3 GHz
Number of cores	16/processor
Memory	1TB/node
Interaction between nodes	40 Gbps QDR InfiniBand
Operating system	Ubuntu 14.04.4
Python version	Python 2.7.13
UPPAAL version	64-bit Linux version 4.1.20
Other libraries	Open MPI 1.6.5, mpi4py 3.0.0

The cluster consists of four computing nodes that are connected by a 40 Gbps InfiniBand network. Each node is allocated $16 \times 4 = 64$ independent processor cores to execute model-based optimization in parallel. Because model-checking operations normally require a huge amount of memory, each node is equipped with 1TB of exclusive memory space.

The Python optimizer runs in Ubuntu Linux operating system, splitting and merging the data across nodes via message passing interface (MPI). Figure 9 depicts the hierarchical structure of the model-based optimizer, which is comprised of the following four modules:

- Input manager that receives the configuration of the optimizer from users and keeps a record of the configuration in an Initialization (INI) file. A configuration class is defined to manage the configuration data including the termination condition, relevant path, SMC and CMC queries, configuration of genetic operators, etc. The INI configuration file is accessed by invoking the Python library ConfigParser.
- Model generator that creates a TA model of the system for each valid parameter vector in the population during the genetic evolution. The system models are described as XML (eXtensible Markup Language) files and generated on the basis of the TA modeling framework introduced in Section 4. The modeling framework further uses the XML Document Object Model (DOM) to read, manipulate, and build an XML model file.
- Genetic algorithm that includes a set of genetic operators (selection, crossover, and mutation) and the fitness function, both of which realize the parallel processing based on MPI. The MPI services are provided by the library Open MPI and accessed by calling the Python package mpi4py. For each individual, the fitness function invokes the UPPAAL command-line verifier Verifyta to evaluate the schedulability and processor occupancy of the corresponding system model.
- Output manager that performs a statistical analysis for each population during the evolution, output the statistical result upon completion of each generation, and provides the final solution after the evolution. The Python library Openpyxl is invoked in order to fill in an Excel table with the output results.

Figure 9. Hierarchical structure of the model-based optimizer.

As shown in Table 2, the optimizer realizes four operator combinations GA 1~ GA 4 of the parallel genetic algorithm.

Table 2. Operator combinations of parallel genetic algorithm.

No.	Crossover Operator	Mutation Operator	Selection Operator
GA 1	Uniform crossover	Bit-flip mutation	Roulette wheel selection
GA 2	Uniform crossover	Bit-flip mutation	Tournament selection
GA 3	Intermediate recombination	Gaussian mutation	Tournament selection
GA 4	Intermediate recombination	Gaussian mutation	Truncation selection

The definition of the genetic operators in Table 2 can be found in [24]. GA 1 and GA 2 adopt binary-valued crossover/mutation operators, together with roulette wheel and tournament selection operator. By contrast, GA 3 and GA 4 use real-valued crossover/mutation operators together with the tournament and truncation selection operator.

6.2. Optimization Experiments

We now carry out the experiments that optimize two different IMA partition systems presented in [9]. Either of the system contains two partitions with the same task set, making the partition priority ordering irrelevant. Both the partitions and tasks use a smaller value to denote a higher priority. In the following two optimization experiments, we assume the overhead of a partition context switch to be two time units.

6.2.1. Experiment 1

The first optimization experiment is performed for the task set of the IMA system shown in Table 3. The optimizer searches for partition periods in the range [5, 100]. For each parameter of a solution, the search precision is defined as 0.5 time unit.

Table 3. IMA task set of Experiment 1.

Partition	Priority	Task				
		Task ID	Period	Execution Time	Deadline	Task Priority
$\mathcal{P}_1$	1	Tsk_1^1	50	5	50	1
		Tsk_2^1	125	7	125	2
		Tsk_3^1	300	6	300	3
$\mathcal{P}_2$	2	Tsk_1^2	50	5	50	1
		Tsk_2^2	125	7	125	2
		Tsk_3^2	300	6	300	3

The parallel genetic algorithm applies the following configuration: the maximum scaling factor of intermediate recombination $d = 0.5$, the bit mutation probability of bit-flip mutation $p_m = 0.2$, the standard deviation of Gaussian mutation $\sigma = 20$, the tournament size of tournament selection $Tour = 2$, the truncation threshold of truncation selection $Trunc = 50\%$, the maximum generation of evolution $G = 100$, and the population size $X = 64$.

UPPAAL model checker is invoked by using the search order “breadth first”, the time limit of SMC simulation $M = 1.0 \times 10^5$, the probability of hypothesis testing $\theta = 0.999$, the number of simulation runs $N = 100$, and the scaling factor of expectation evaluation $K = 1.0 \times 10^4$.

Table 4 presents the optimization result of Experiment 1. GA 3 and GA 4 both acquired the same optimal solution as that of an exhaustive search, reaching the lowest processor occupancy 46%. By contrast, the output solutions of GA 1 and GA 2 have slightly higher processor occupancy (48.3% and 51.66%, respectively) than that of either GA 3 or GA 4.

Table 4. Optimization result of Experiment 1.

No.	Solution	Processor Occupancy	Optimal
GA 1	(51.5, 10.5, 52, 10.5)	48.3%	No
GA 2	(52.5, 12, 54, 11.5)	51.66%	No
GA 3	(50, 9.5, 50, 9.5)	46%	Yes
GA 4	(50, 9.5, 50, 9.5)	46%	Yes

6.2.2. Experiment 2

Table 5 shows the IMA task set of the second optimization experiment, where we keep the configuration of Experiment 1, except the search range of partition periods, the maximum generation of evolution, and the time limit of SMC simulation. Consider the larger periods and deadlines in the second IMA system. The optimizer searches for partition periods in a larger range [5, 200]. The parallel genetic algorithm repeats until the new maximum generation $G = 200$ is reached. UPPAAL also uses a double time limit of SMC simulation $M = 2.0 \times 10^5$.

Table 5. IMA task set of Experiment 2.

Partition	Priority	Task				
		Task ID	Period	Execution Time	Deadline	Task Priority
$\mathcal{P}_1$	1	Tsk_1^1	160	8	100	1
		Tsk_2^1	240	12	200	2
		Tsk_3^1	320	16	300	3
		Tsk_4^1	480	24	400	4
$\mathcal{P}_2$	2	Tsk_1^2	160	8	100	1
		Tsk_2^2	240	12	200	2
		Tsk_3^2	320	16	300	3
		Tsk_4^2	480	24	400	4

Table 6 shows the optimization result of Experiment 2. GA 3 and GA 4 still reached the globally optimal solution with the processor occupancy 45.62%. When compared with Experiment 1, GA 1 and GA 2 obviously deviated from reasonable search areas of the parameter space. Finally, their output solutions obtained much higher processor occupancy (75.09% and 78.1%, respectively) than that of GA 3 and GA 4.

Table 6. Optimization result of Experiment 2.

No.	Solution	Processor Occupancy	Optimal
GA 1	(159.5, 53.5, 165, 64.5)	75.09%	No
GA 2	(160.5, 48.5, 156.5, 69)	78.1%	No
GA 3	(160, 34.5, 160, 34.5)	45.62%	Yes
GA 4	(160, 34.5, 160, 34.5)	45.62%	Yes

6.2.3. Discussion

In both experiments, the operator combinations GA 3 and GA 4 generated the same optimal solution as the exhaustive search and finally reached the minimum processor occupancy. However, the exhaustive search spent 42 and 125 days on the system of Experiment 1 and 2, respectively. It is obviously infeasible to cope with a concrete avionics system via exhaustive search. In contrast, the operator combinations GA 1 and GA 2 failed to find the optimal parameters or approached the minimum processor occupancy. Moreover, the processor occupancy gaps between GA 1/GA 2 and GA 3/GA 4 are further widened as the solution space increases from Experiment 1 to 2. Thus, we conclude that the combination of intermediate recombination and Gaussian mutation operator is more suitable for the parameter optimization of IMA partition scheduling systems and it is able to find parameters with better processor occupancy.

Figure 10 shows the evolution of the optimal processor occupancy of populations in the experiments. It can be observed that the operator combinations GA 1 and GA 2 in both experiments converged on a locally optimal solution prematurely due to the low search efficiency of the binary-valued crossover and mutation operators. During the evolution process, GA 1 and GA 2 only generated few offspring individuals with better fitness randomly, but these few individuals quickly took up a dominant proportion of the following populations, leading to a rapid decrease in the degree of population diversity. After the 100th generation, the populations lost their evolutionary capability, eventually converging on random locally-optimal solutions. This premature convergence has a substantially adverse effect on Experiment 2 that defines a larger parameter space, which makes it difficult for GA 1 or GA 2 to optimize the processor occupancy of the system.

Figure 10. Evolution of minimum processor occupancy in optimization experiments. (a) Experiment 1. (b) Experiment 2.

By contrast, the real-valued crossover and mutation operators of GA 3 and GA 4 have better search efficiency to produce high-fitness individuals continuously, finally reaching a globally optimal solution in both experiments. GA 3 and GA 4 share the same configuration except different selection operators, which still lead to the distinct evolution patterns of processor occupancy in Figure 10. Because the tournament selection of GA 3 has higher selective pressure than the truncation selection of GA 4, GA 3 achieved faster convergence than GA 4, but this also increased the risk of premature convergence. In practice, two selection operators can be used in combination to perform the optimization, thereby keeping a balance between selection pressure and population diversity.

6.3. Comparison Experiment

In this section, the proposed model-based method is compared with analytical optimization methods to demonstrate the superior performance of our method. This comparison experiment optimizes the same two avionics systems as Section 6.2 by using the following four optimization methods:

1. Geometric Programming (GP): as described in [4], the GP method establishes the schedulability constraints of the system according to a temporal demand and supply analysis, formulating the optimization problem as a geometric programming model. We implement the GP optimizer in Python 3.8, where the library Gpkit 0.9.9 is used to solve the GP problem.
2. Heuristic Search based on Response Time Analysis (HS-RTA): the HS-RTA method [9] adopts a more precise response time analysis to model the schedulability condition of the system and performs a heuristic search of unknown parameter space in accordance with an empirical investigation, finally providing the best solution it has found.
3. Model-Based Method (MBM) proposed in this paper: because the operator combinations GA 3 and GA 4 have been proved feasible for the typical IMA systems, we apply the operator combination GA 4 together with the same configuration as that in Experiment 1 and 2.
4. Exhaustive Search (ES): this method checks the models of all possible parameter combinations with the specific precision [14], thus obtaining the globally optimal solution with the minimum processor occupancy. Although ES is infeasible for the concrete systems due to its extremely long processing time, it is still used to provide this experiment with a baseline for processor occupancy for comparison. The MBM and ES method both apply the same search precision 0.5 time unit to this comparison experiment.

Figure 11 presents the processor occupancy of the output solutions of the above four optimization methods in this comparison experiment. The workload of a system is defined as the sum of the

execution time/period ratios of all tasks. System 2 has a heavier workload 40% than System 1 with 35.2%, as shown in Figure 11.

Figure 11. Processor occupancy of output solutions in comparison experiment. (a) System 1 with workload 35.2%. (b) System 2 with workload 40%.

It can be observed that two analytical methods produced conservative optimization results in this comparison experiment. Their solutions have higher processor occupancy than those of the model-based methods.

The GP model requires the schedulability condition to be formulated as the constraints on posynomial and monomial expressions. The method introduces conservative worst-case assumptions into the temporal demand and supply analysis to construct the mathematical definition of GP, thus leading to the pessimistic processor occupancy of its output solutions.

By contrast, the HS-RTA method is able to generate better solutions than GP owing to the exact response time analysis. However, HS-RTA still lagged behind the model-based methods in the final processor occupancy. As the workload increased from 35.2% of System 1 to 40% of System 2, there was a wider gap between the analytical and model-based methods.

The model-based methods benefit from the powerful expressiveness of timed automaton models, thereby providing a precise description of the systems. Moreover, our model-based method MBM outperformed the analytical ones and reached the same globally-optimal processor occupancy as ES within 200 generations.

7. Conclusions

The partition scheduling of IMA systems presents a challenge to traditional analytical methods, which introduce various oversimplifications into this complex nonlinear non-convex optimization problem, thus leading to unnecessary waste of processor time. We conclude that the model-based approach proposed in this paper is applicable to the optimization of IMA partition scheduling. The expressive timed automaton models provide a precise description of an IMA partition scheduling system and more accurate schedulability bounds in the optimization problem. The case studies demonstrate that our model-based method outperforms the representative analytical methods on the processor occupancy of IMA partition scheduling systems.

The model-based optimizer also integrates a parallel genetic algorithm and statistical model checking (SMC) in order to speed up the optimization process. The parallel genetic algorithm only explores the promising solutions instead of an exhaustive search and distributes the costly model-based analyses across nodes in a cluster. The application of SMC not only fast falsifies the schedulability of promising solutions but also precisely evaluates their processor occupancy. The optimization

experiments show that our model-based method is able to achieve the same globally-optimal processor occupancy as an exhaustive search within an acceptable processing time.

The model-based method also introduces limitations into this optimization approach. The main bottleneck of our approach is the processing of classical model checking (CMC) in the last step. When handling larger system models, the engine of CMC may not finish the schedulability verification within an acceptable time due to the well-known state-space explosion problem. As future work, we plan to apply the compositional model-based method to more complex avionics systems in order to mitigate the state-space explosion and design a dedicated evolutionary algorithm that further improves search efficiency.

Author Contributions: Conceptualization, P.H. and Z.Z.; methodology, P.H.; software, P.H.; validation, P.H. and L.Z.; formal analysis, P.H.; investigation, P.H.; resources, P.H.; data curation, L.Z.; writing—original draft preparation, P.H.; writing—review and editing, Z.Z.; visualization, P.H.; supervision, Z.Z.; project administration, Z.Z.; funding acquisition, Z.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China under grant number 61601371 and the Aviation Science Fund of China under grant number 2016ZD53035.

Acknowledgments: The authors would like to thank Brian Nielsen and Ulrik Nyman of Aalborg University for their great help on conceptualization and technical support for the model checker UPPAAL.

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

References

1. Bieber, P.; Boniol, F.; Boyer, M.; Noulard, E.; Pagetti, C. New Challenges for Future Avionic Architectures. *Aerospace Lab J.* **2012**, *4*, 1–10.
2. AEEC. *Avionics Application Software Standard Interface: Part 1—Required Services*; ARINC Specification 653P1-5; Aeronautical Radio Inc.: Cedar Rapids, IA, USA, 2019.
3. Carnevali, L.; Pinzuti, A.; Vicario, E. Compositional verification for hierarchical scheduling of real-time systems. *IEEE Trans. Softw. Eng.* **2013**, *39*, 638–657. [[CrossRef](#)]
4. Yoon, M.K.; Kim, J.E.; Bradford, R.; Sha, L. Holistic design parameter optimization of multiple periodic resources in hierarchical scheduling. In Proceedings of the Conference on Design, Automation and Test in Europe (EDA Consortium), Grenoble, France, 13 March 2013; pp. 1313–1318.
5. Lipari, G.; Bini, E. A methodology for designing hierarchical scheduling systems. *J. Embed. Comput.* **2005**, *1*, 257–269.
6. Shin, I.; Lee, I. Compositional real-time scheduling framework with periodic model. *ACM Trans. Embed. Comput. Syst. (TECS)* **2008**, *7*, 30. [[CrossRef](#)]
7. Easwaran, A.; Lee, I.; Sokolsky, O.; Vestal, S. A compositional scheduling framework for digital avionics systems. In Proceedings of the 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (ERCSA 2009), Beijing, China, 24–26 August 2009.
8. Dewan, F.; Fisher, N. Approximate bandwidth allocation for fixed-priority-scheduled periodic resources. In Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium, Stockholm, Sweden, 12–15 April 2010; pp. 247–256.
9. Davis, R.; Burns, A. An investigation into server parameter selection for hierarchical fixed priority pre-emptive systems. In Proceedings of the 16th International Conference on Real-Time and Network Systems (RTNS 2008), Rennes, France, 16–17 October 2008.
10. Blikstad, M.; Karlsson, E.; Löw, T.; Rönnerberg, E. An optimisation approach for pre-runtime scheduling of tasks and communication in an integrated modular avionic system. *Optim. Eng.* **2018**, *19*, 977–1004. [[CrossRef](#)]
11. Mikučionis, M.; Larsen, K.G.; Rasmussen, J.I.; Nielsen, B.; Skou, A.; Palm, S.U.; Pedersen, J.S.; Hougaard, P. Schedulability Analysis Using Uppaal: Herschel-Planck Case Study. In *Leveraging Applications of Formal Methods, Verification, and Validation*; Margaria, T.; Steffen, B., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; pp. 175–190.

12. Cicirelli, F.; Furfaro, A.; Nigro, L.; Pupo, F. Development of a schedulability analysis framework based on pTPN and UPPAAL with stopwatches. In Proceedings of the 2012 IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications, Dublin, Ireland, 25–27 October 2012.
13. Beji, S.; Hamadou, S.; Gherbi, A.; Mullins, J. SMT-based cost optimization approach for the integration of avionic functions in IMA and TTEthernet architectures. In Proceedings of the 2014 IEEE/ACM 18th International Symposium on Distributed Simulation and Real Time Applications, Toulouse, France, 1–3 October 2014; pp. 165–174.
14. Sun, Y.; Lipari, G.; Soulat, R.; Fribourg, L.; Markey, N. Component-based analysis of hierarchical scheduling using linear hybrid automata. In Proceedings of the 2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications, Chongqing, China, 20–22 August 2014.
15. Boudjadar, J.; Larsen, K.G.; Kim, J.H.; Nyman, U. Compositional schedulability analysis of an avionics system using UPPAAL. In Proceedings of the ICAASE 2014 (International Conference on Advanced Aspects of Software Engineering), Constantine, Algeria, 2–4 November 2014.
16. Kim, J.H.; Legay, A.; Traonouez, L.M.; Boudjadar, A.; Nyman, U.; Larsen, K.G.; Lee, I.; Choi, J.Y. Optimizing the resource requirements of hierarchical scheduling systems. *ACM Sigbed Rev.* **2016**, *13*, 41–48. [CrossRef]
17. Boudjadar, A.; David, A.; Kim, J.H.; Larsen, K.G.; Mikučionis, M.; Nyman, U.; Skou, A. Statistical and exact schedulability analysis of hierarchical scheduling systems. *Sci. Comput. Program.* **2016**, *127*, 103–130. [CrossRef]
18. Han, P.; Zhai, Z.; Nielsen, B.; Nyman, U.; Kristjansen, M. Schedulability Analysis of Distributed Multicore Avionics Systems with UPPAAL. *J. Aerosp. Inf. Syst.* **2019**, *16*, 473–499. [CrossRef]
19. Annighöfer, B.; Kleemann, E.; Thielecke, F. Model-based Development of Integrated Modular Avionics Architectures on Aircraft-level. In Proceedings of the Deutscher Luft-und Raumfahrtkongress, Bremen, Germany, 27–29 September 2011.
20. Behrmann, G.; David, A.; Larsen, K.G. A tutorial on UPPAAL. In *Formal Methods for the Design of Real-Time Systems*; Springer: New York, NY, USA, 2004; pp. 200–236.
21. David, A.; Larsen, K.G.; Legay, A.; Mikučionis, M.; Poulsen, D.B. UPPAAL SMC tutorial. *Int. J. Softw. Tools Technol. Transf.* **2015**, *17*, 397–415. [CrossRef]
22. UPPAAL Home. Available online: <http://www.uppaal.org/> (accessed on 20 June 2020).
23. Han, P.; Zhai, Z.; Nielsen, B.; Nyman, U. A Modeling Framework for Schedulability Analysis of Distributed Avionics Systems. In Proceedings of the 3rd Workshop on Models for Formal Analysis of Real Systems and 6th International Workshop on Verification and Program Transformation, Thessaloniki, Greece, 20 April 2018; pp. 150–168. [CrossRef]
24. Bäck, T.; Fogel, D.B.; Michalewicz, Z. *Evolutionary Computation 1—Basic Algorithms and Operators*; CRC Press: Boca Raton, FL, USA, 2018.

© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).