mathematics

MDPI

Article

Contention-Free Scheduling for Single Preemption Multiprocessor Platforms

Hyeongboo Baek [1] and Jaewoo Lee [2,*]

[1] Department of Computer Science and Engineering, Incheon National University, Incheon 22012, Republic of Korea; hbbaek@inu.ac.kr
[2] Department of Industrial Security, Chung-Ang University, Seoul 06974, Republic of Korea
* Correspondence: jaewoolee@cau.ac.kr; Tel.: +82-2-820-5935

Abstract: The Contention-Free (CF) policy has been extensively researched in the realm of real-time multi-processor scheduling due to its wide applicability and the performance enhancement benefits it provides to existing scheduling algorithms. The CF policy improves the feasibility of executing other real-time tasks by assigning the lowest priority to a task at a moment when it is guaranteed not to miss its deadline during the remaining execution time. Despite its effectiveness, existing studies on the CF policy are largely confined to preemptive scheduling, leaving the efficiency and applicability of limited preemption scheduling unexplored. Limited preemption scheduling permits a job to execute to completion with a limited number of preemptions, setting it apart from preemptive scheduling. This type of scheduling is crucial when preemption or migration overheads are either excessively large or unpredictable. In this paper, we introduce SP-CF, a single preemption scheduling approach that incorporates the CF policy. SP-CF allows a preemption only once during each job's execution, following a priority demotion under the CF policy. We also propose a new schedulability analysis method for SP-CF to determine whether each task is executed in a timely manner and without missing its deadline. Through simulation experiments, we demonstrate that SP-CF can significantly enhance the schedulability of the traditional rate-monotonic algorithm and the earliest deadline first algorithm.

Keywords: real-time multi-processor scheduling; contention-free policy; limited preemption scheduling

MSC: 68M20

check for updates

Citation: Baek, H.; Lee, J. Contention-Free Scheduling for Single Preemption Multiprocessor Platforms. *Mathematics* **2023**, *11*, 3547. https://doi.org/10.3390/math11163547

Academic Editors: Chin-Chia Wu, Win-Chin Lin, Jatinder N. D. Gupta and Xingong Zhang

Received: 11 July 2023
Revised: 13 August 2023
Accepted: 15 August 2023
Published: 16 August 2023

Copyright: © 2023 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 (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Real-time systems are characterized not only by their functional traits but also by their temporal ones [1]. For instance, in automotive systems, the wheel control system's precise execution within a set timeframe, or deadline, is vital for accurate vehicle control. A key technique in designing such systems is real-time scheduling, which allows for the efficient distribution of computing resources like CPU and memory within the system. Real-time scheduling research primarily focuses on two critical areas: the design of scheduling algorithms and schedulability analysis. The former determines the execution order of real-time tasks, while the latter involves a mathematical assessment of whether a system can meet its deadlines under a scheduling algorithm [2–5].

In recent decades, real-time systems have made effective use of multiprocessor platforms to execute tasks with substantial computational demands [6]. This trend has spurred extensive theoretical research in real-time scheduling. Some studies have introduced new real-time scheduling algorithms that effectively alter task priorities during runtime [7], while others have concentrated on theoretical optimality or clarifying the relationships between different scheduling algorithms [8]. Additional research has proposed execution techniques that can be incorporated with existing scheduling algorithms [9–11].

Emphasizing the latter approach is vital, given its potential to enhance both current and future algorithm scheduling performance. The Zero-Laxity (ZL) [9] policy and the Contention-Free (CF) policy [10,11] are successful examples of this approach. The ZL policy reduces deadline violations by assigning the highest priority to a real-time task at an instant where the task would fail to meet its deadline if not immediately executed. Conversely, the CF policy enhances the feasibility of executing other real-time tasks by assigning the lowest priority to a task at a moment when it is guaranteed not to miss its deadline during the remaining execution time [12–14].

The CF policy has been extensively studied in the field of scheduling due to their broad applicability and the performance enhancement benefits they offer to existing scheduling algorithms. However, existing studies are largely confined to preemptive scheduling, leaving the efficiency and applicability of limited preemption scheduling unexplored. Limited preemption scheduling permits a job to execute to completion with a limited number of preemptions, distinguishing it from preemptive scheduling. This scheduling type is essential when preemption or migration overheads are either excessively large or unpredictable.

In this paper, we introduce SP-CF, a single preemption scheduling approach that incorporates the CF policy. SP-CF allows a preemption only once during each job's execution, following a priority demotion under the CF policy. We also propose a new schedulability analysis method for SP-CF to judge whether each task is executed timely and without missing its deadline. The performance of this technique is evaluated using simulation-based experiments.

Section 3 discusses the system model and basic concepts targeted in this study. Section 4 presents the operation of the single-preemption scheduling with the CF policy. In Section 5, we extend the existing schedulability analysis called deadline-based (DA) analysis [15] to our SP-CF scheduling. Section 6 uses our Java simulator to analyze the performance of DA analysis, and Section 2 presents the related work. Section 8 provides the paper's conclusion.

2. Related Work

In the past few decades, numerous research studies within the realm of real-time systems have been centered on creating scheduling algorithms for multiprocessor systems. The aim is to enhance the system's schedulability. Notably, ER-Fair [16], LLREF [17], and RUN [7] have been introduced as the optimal scheduling algorithms for multiprocessor systems, with each iteration refining the scheduling costs [17]. A salient example is the RUN algorithm, which repurposes the multiprocessor task scheduling problem into a uniprocessor issue, thereby significantly reducing the amount of preemptions when compared to rate-based methodologies.

Simultaneously, an alternate approach has been pursued that involves the creation of scheduling policies designed to enhance schedulability analysis and can be integrated into most existing scheduling algorithms. The ZL and CF policies are prominent examples of such strategies. The ZL policy [9] boosts a task's priority to the highest level when it is scheduled to prevent any deadline misses, while the CF policy [10,11] reduces the priority to the lowest level when its schedulability is secured. Additional research into the CF policy was conducted to better utilize contention-free slots [18]. Furthermore, a proposal was put forth for RTA associated with the CF policy.

Numerous studies have been conducted on non-preemptive scheduling for uniprocessor systems. Ekelin presented a scheduling algorithm designed for a set of independent jobs [19]. Meanwhile, Nasri et al. proposed a scheduling algorithm specifically for the periodic (loose-) harmonic task model [20,21]. This model was later improved to support the general periodic task model [22]. Furthermore, efforts have been made to develop new types of schedulability tests that provide timing guarantees for existing scheduling approaches [23,24]. The research has also extended beyond uniprocessor systems. One study examined the scheduling of mixed-criticality tasks [25], while another research effort fo-

cused on scheduling within a multiprocessor environment, which necessitated information about future job release patterns [6].

3. System Model

In this paper, we adopt the Liu & Layland task model [1], widely used in real-time systems. As per this model, a system is comprised of a task set τ, containing n tasks, denoted $\tau_i \in \tau$. Each task τ_i periodically generates an infinite number of jobs J_i. A task τ_i is represented as (T_i, C_i, D_i), where T_i is the task's period, C_i signifies the worst-case execution time of a job, and D_i indicates the time interval between a job's release time and its deadline.

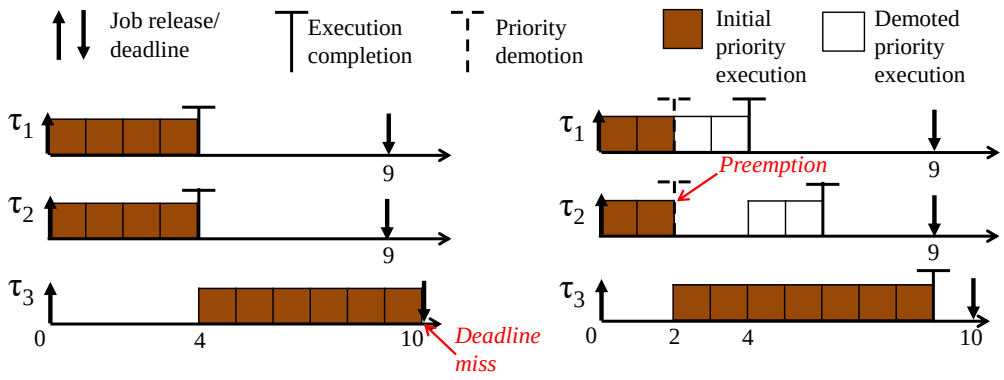
This paper assumes an m multiprocessor environment. A time slot is defined as the minimum unit of time in which a job can be executed, with each job restricted to execute on a single processor within a time slot. We consider a fixed-priority scheduling and earliest deadline first (EDF) algorithm. The former assigns priorities to tasks themselves, where all jobs generated by a task share the same priority. Accordingly, all jobs generated by tasks with higher priority precede those generated by tasks with lower priority in terms of priority. The latter assigns priorities to jobs rather than tasks, where a job with earlier absolute deadline has a higher priority. A set of tasks with a higher priority than τ is denoted as τ_{hp}, and a set of tasks with lower priority than τ is denoted as τ_{lp}. A CF slot is characterized as a time slot where the number of available jobs is fewer than the number of processors m. Thus, during idle time slots, all jobs can execute without competition. According to SP-CF policy, preemption occurs only once during each job's execution.

4. Single Preemption Scheduling with Cf Policy

In this section, we present the background of the fixed-priority non-preemptive scheduling algorithm with the CF policy and the calculation method for CF slots.

4.1. Scheduling

Firstly, we describe the scheduling patterns of both non-preemptive scheduling and SP-CF scheduling using an example task set. Figure 1a depicts the scheduling pattern when three tasks, $\tau_1 = (15, 4, 9)$, $\tau_2 = (15, 4, 9)$, and $\tau_3 = (15, 7, 10)\}$, are executed with the fixed-priority scheduling algorithm in a two-processor system. Figure 1b illustrates the scheduling pattern when applying the CF policy to the scheduling algorithm. Here, each job is allowed to preempt only once following a priority demotion according to the SP-CF policy. There are numerous methods to assign priorities to tasks within the fixed-priority scheduling algorithm. For this example, we assume that task τ_1 has the highest priority, task τ_2 holds intermediate priority, and task τ_3 has the lowest priority.



(a) Non-preemptive Fixed-priority scheduling (b) SP-CF scheduling

Figure 1. Scheduling scenario of non-preemptive fixed-priority scheduling and SP-CF scheduling for three tasks $\tau_1 = (15, 4, 9)$, $\tau_2 = (15, 4, 9)$, and $\tau_3 = (15, 7, 10)\}$ on a two-processor system.

As seen in Figure 1a, tasks τ_1 and τ_2 are executed on individual processors in the time slots from 0 to 4. Upon completion of these two tasks, task τ_3 begins execution. However, the time slot required for task τ_3 to complete its execution exceeds the remaining time slots, resulting in a deadline violation at time 10.

Figure 1b portrays a scenario where the application of the CF policy to the fixed-priority scheduling algorithm ensures all tasks can execute without missing a deadline. Initially, the CF policy calculates the minimum number of CF slots existing within each task's job's release time and deadline (the method for calculating the number of CF slots will be detailed in Section 4.2). This calculation reveals at least two CF slots between the release time and deadline of tasks τ_1 and τ_2. Between the time slots 0 and 4, there are three runnable jobs and two processors, creating a need for contention slots. According to the CF policy, at time 2, tasks τ_1 and τ_2 receive the lowest priority, and task τ_3 commences its execution. This is because the calculation shows that tasks τ_1 and τ_2 have at least two CF slots. The lack of contention slots between 0 and 2 implies that the remaining executions can take place in CF slots, ensuring no deadline misses. Thus, by comparing the remaining execution time with the residual CF slots, it is feasible to dynamically lower the priority and efficiently utilize the computational power of the multiprocessor. Please note that a preemption occurs only for τ_2 at time 2, following a priority demotion by the CF policy.

As depicted in Figure 1a, the duration when both processors are occupied is constrained to 4 units. However, as demonstrated in Figure 1b, when implementing the CF policy, the utilization period of both processors extends to 6 units.

Algorithm 1 describes the execution procedure of the SP-CF scheduling. For each time slot at t, a job J_i released by a task τ_i is placed in Q^{H}. The corresponding number of CF slots Φ_i for the job is calculated, and the remaining execution time $C_i(t)$ and remaining CF slots $\Phi_i(t)$ are initialized with C_i and $\Phi_i(t)$, respectively (lines 1–3).

Algorithm 1 SP-CF scheduling.

At each time slot t,
if a job from τ_i is released **then**
Place the job into Q^{H} and initialize $\Phi_i(t) \leftarrow \Phi_i$ and $C_i(t) \leftarrow C_i$.
end if
for each job in Q^{H} **do**
if the job from τ_i satisfies $\Phi_i(t) \geq C_i(t)$ **then**
Transfer the job to Q^{L}.
end if
end for
if $|Q^{\mathrm{H}}|$ is less or equal to m **then**
Update $\Phi_i(t+1) \leftarrow \max\left(0, \Phi_i(t) - 1\right)$ for all jobs in Q^{H}
end if
Assign priorities to jobs within the following three groups: hp, mp, and lp. Group hp has a strictly higher priority than mp, and mp has a strictly higher priority than lp. Jobs of hp were executed at time $t-1$ and did not transition from Q^{H} to Q^{L} at time t. Jobs of mp were released at time t and are currently in Q^{H}. Jobs of lp that transitioned from Q^{H} to Q^{L} at time t or were present in Q^{L} but were not executed at time $t-1$.
Assign priorities to jobs released at t in mp according to the base algorithms such as EDF and RM.
Decrease $C_i(t) \leftarrow C_i(t) - 1$ for the (at most) m highest-priority jobs in hp, mp, and lp.

If a job J_i exists in a time slot at t with a remaining execution time $C_i(t)$ less than or equal to $\Phi_i(t)$, the job is moved to Q^{L} (lines 4–8). For each job in Q^{H}, if the current time slot is a CF slot, $\Phi_i(t)$ decreases by 1 (lines 9–11). Priorities are then assigned to the jobs in Q^{H} based on a fixed priority algorithm (line 12). Finally, at most m jobs in Q^{H} and Q^{L} are executed, where all jobs in Q^{H} hold a strictly higher priority than all jobs in Q^{L} (line 13).

Priorities are assigned within the following three groups: hp, mp, and lp. The hp group has a strictly higher priority than mp, and mp has a strictly higher priority than lp.

hp: jobs that were executed at time $t-1$ and did not transition from Q^{H} to Q^{L} at time t.
mp: jobs that were released at time t and are currently in Q^{H}.
lp: jobs that transitioned from Q^{H} to Q^{L} at time t or were present in Q^{L} but were not executed at time $t-1$.

The following lemma discusses a specific property of SP-CF scheduling concerning the number of preemptions for each job.

Lemma 1. *For a given set of real-time tasks on m processors, each job experiences at most one preemption under SP-CF scheduling.*

Proof. Let us suppose a job of τ_i can experience multiple preemptions. From line 12 in Algorithm 1, it can be observed that running jobs (i.e., jobs executed at time $t-1$ in hp) that do not experience a priority demotion (i.e., jobs that do not transition from Q^{H} to Q^{L}) at time t holds a highest priority compared to jobs in mp and lp. Moreover, this priority can only be altered when a job shifts from hp or mp to lp after experiencing a priority demotion, thereby inducing a preemption possibility. This contradicts our initial assumption for the lemma. □

4.2. Contention-Free Slots

As observed in Algorithm 1, the CF policy calculates the minimum number of CF slots that can exist before the deadline of each job (line 2). When the remaining execution time and remaining CF slots of a job become equal during execution, the job's priority is demoted to the lowest (i.e., moved to Q^{L}). Jobs with demoted priority can encounter at least as many CF slots as the remaining execution time, ensuring that no deadline miss occurs. In this section, we will explore how to calculate the number of CF slots for each task.

In real-time scheduling with the CF policy, the number of CF slots for each task is precalculated before execution, and its value decreases whenever a CF slot is encountered during execution. The fundamental concept used in calculating the number of CF slots is to analyze the execution loads of all possible jobs between the release time and the deadline of a single job. Afterward, we determine the maximum number of contention slots (i.e., time slots where competition occurs, and the number of running jobs exceeds the number of processors) that the jobs need to compete for processor usage. This value is referred to as the contention slot value. By subtracting the contention slot value from the existing time slots, we can calculate the minimum number of CF slots.

To determine the minimum number of CF slots between the release time and the deadline of job generated by τ_i, we first need to calculate the maximum possible execution load within that interval.

Figure 2 illustrates the maximum achievable execution load of within the interval between the release time and the deadline [26]. As seen in Figure 2, in the scenario where the maximum execution of occurs, the execution of the first job starts at the beginning of the interval L and ends at the deadline of that job. The subsequent jobs are executed as soon as they are generated to maximize the amount of execution. The maximum achievable execution load of within the interval is calculated as the sum of the number of running jobs that can be executed and the execution load of jobs that cannot be executed.

Lemma 2 (Execution load of task τ_i)**.** *The maximum execution load of a task τ_i within the interval L, denoted as $W_i(L)$, can be calculated as follows:*

$$W_i(L) = n_i(L) \cdot C_i + \min(C_i, L + D_i - C_i - n_i(L) \cdot T_i)) \tag{1}$$

where $n_i(L) = \left\lfloor \frac{L+D_i-C_i}{T_i} \right\rfloor$ represents the total number of jobs that execute fully for C_i within the interval L, and whose subsequent job release also exists within the interval L.

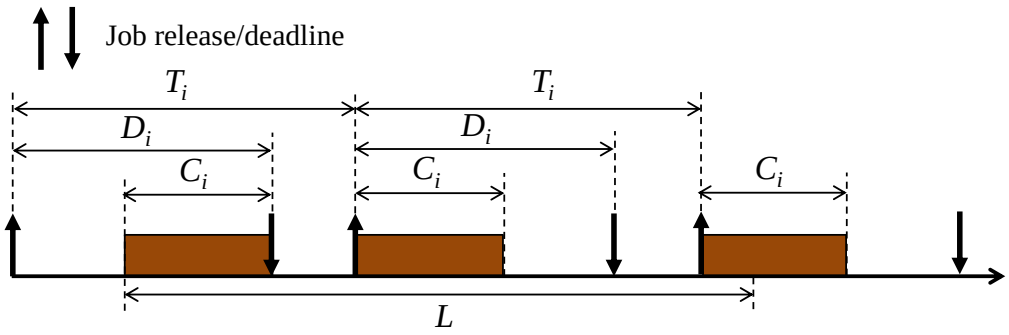


Figure 2. Workload under any scheduling in an interval of length L.

Proof. By the definition of $n_i(L)$, at most $n_i(L)$ jobs can fully execute for C_i units within the interval L. The rest of the time in L, $L + D_i - C_i - n_i(L) \cdot T_i$, can accommodate the execution of at most one more job. Therefore, the maximum execution load $W_i(L)$ is given by Equation (1). □

For instance, in Figure 1a, from τ_3's perspective, the deadlines of the first jobs of τ_1 and τ_2 come before the end of the interval of interest, which is the D_3 interval of τ_3. Therefore, these jobs belong to $n_3(L)$. Note that W_i represents the maximum workload that can occur in the worst-case scenario, so the sum of the higher priority executions in the example of Figure 1a, which depicts a real case, can be less than the maximum workload. Using the maximum execution load W_i, we can calculate the minimum number of CF slots that can exist in the interval L. Assuming m processors, the size of the available execution time space in the interval is $m \cdot L$. In each contention time slot, at least m tasks execute. Therefore, the number of contention slots that can exist in the interval L cannot exceed the following value.

Lemma 3 (Contention slots [11])**.** *The number of contention slots that can exist in the interval L does not exceed the following value:*

$$\left\lfloor \frac{\sum_{\tau_i \in \tau} W_i(L)}{m} \right\rfloor \tag{2}$$

Proof. Given that each contention slot accommodates at least m tasks, the sum of the execution loads of all tasks in τ cannot exceed m times the number of contention slots. □

We can use the maximum contention slots obtained from Equation (2) to calculate the minimum CF slots in the interval D_k of τ_k as follows.

Theorem 1 (Minimum CF slots)**.** *The minimum number of CF slots that can exist in the interval D_k of τ_k, denoted as Φ_k, can be calculated as follows:*

$$\Phi_k = \max\left(0, D_k - \left\lfloor \frac{C_k + \sum_{\tau_i \in \tau \setminus \tau_k} W_i(D_k)}{m} \right\rfloor\right) \tag{3}$$

Proof. From Lemma 3, the number of contention slots does not exceed $\left\lfloor \frac{\sum_{\tau_i \in \tau} W_i(D_k)}{m} \right\rfloor$. Since we are looking at the D_k interval, there is necessarily only one job of τ_k within this interval. Therefore, the expression can be reduced to $\left\lfloor \frac{C_k + \sum_{\tau_i \in \tau \setminus \tau_k} W_i(D_k)}{m} \right\rfloor$. Hence, the rest of the slots in D_k must be CF slots. If the calculated number of CF slots is negative, it is corrected to zero. □

For example, in Figure 1b, from τ_3's perspective, the deadlines of the first jobs of τ_1 and τ_2 come before the end of the interval of interest, which is the D_k interval of τ_3. Therefore,

these jobs belong to $n_i(D_k)$. Additionally, the third and fourth executions of these jobs are run after their priorities are demoted, so they belong to Φ_k.

5. Schedulability Analysis under SP-CF Scheduling

In this section, we extend the existing DA analysis proposed for general scheduling to SP-CF scheduling, aiming to propose a new schedulability analysis to judge the timely execution of tasks.

5.1. Da for SP-CF with Fixed-Priority Scheduling

The response time of task τ_i is defined as the biggest difference between the release time and the finish time of all jobs J_i belonging to τ_i. By the definition of response time of task τ_i, if the response time of τ_i is less than or equal to D_i, it is determined that all jobs J_i can complete their execution within its deadline. The following lemma provides the method whether the response time of a task τ_k under SP-CF with fixed-priority scheduling is less than or equal to D_k.

Lemma 4 (DA for fixed-priority scheduling)**.** *Given $W_i(D_k)$ as the maximum execution load of task τ_i within interval D_k on m processors, the task τ_k can complete its execution within D_k if the following equation holds:*

$$\max_{\tau_j \in \tau_{lp(k)}} C_j + \frac{\sum_{\tau_i \in \tau_{hp(k)}} \min(W_i(D_k), D_k - C_k + 1)}{m} + C_k \leq D_k, \tag{4}$$

where $lp(k)$ and $hp(k)$ represent the sets of tasks within τ that have a lower priority and a higher priority than τ_k, respectively.

Proof. Due to the property of single-preemptiveness, the execution of a task τ_k can be blocked by an already executing lower-priority job, and the amount of blocking cannot be larger than $\max_{\tau_j \in \tau_{lp(k)}} C_j$. By Lemma 2, the maximum execution of higher-priority task τ_i in interval D_k is upper-bounded by $W_i(D_k)$. If the value $W_i(D_k)$ exceeds C_k, it implies that the task τ_k can be executed concurrently, and the maximum value of $W_i(D_k)$ is $D_k - C_k + 1$. In a system with m processors, a task experiences interference in a time slot if there are at least m jobs with higher priority. For a job of τ_k to be response, the execution for C_k should be completed. Thus, the lemma holds. □

Then, we derive DA for SP-CF with fixed-priority scheduling. The following lemma derives the execution load of task τ_i under SP-CF with fixed-priority scheduling.

Lemma 5 (Execution load of task τ_i under SP-CF)**.** *The maximum execution load of task τ_i within interval D_k under SP-CF scheduling is given by:*

$$W_i^{\Phi}(D_k) = n_i^{\Phi}(D_k) \cdot (C_i - \Phi_i) + \min(C_i - \Phi_i, D_k + D_i - C_i - \Phi_i - n_i(D_k) \cdot T_i), \tag{5}$$

where $n_i^{\Phi}(D_k)$ is derived by

$$n_i^{\Phi}(D_k) = \left\lfloor \frac{D_k + D_i - C_i - \Phi_i}{T_i} \right\rfloor. \tag{6}$$

Proof. During the execution of task τ_i, there exists a minimum number of CF slots calculated by Equation (3) in Theorem 1. This allows task τ_i to execute without contention in these slots. Hence, for each task τ_i, we can exclude an amount of interference time equal to Φ_i as shown in Figure 3. By Lemma 2, the lemma holds. □

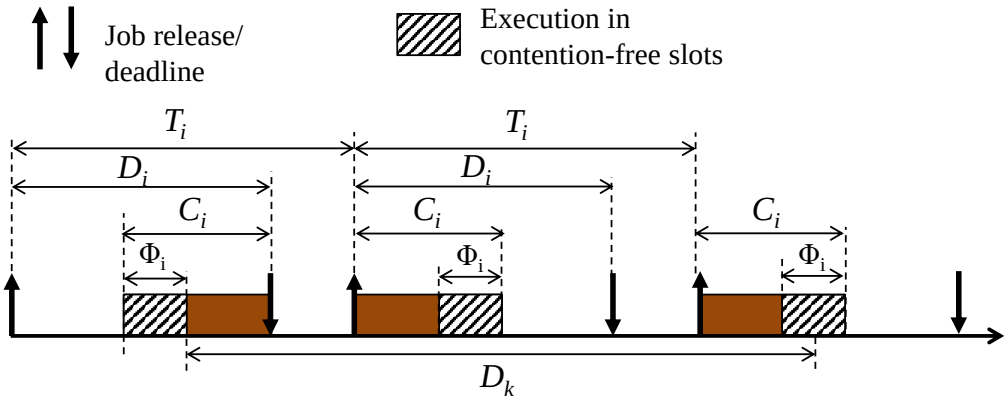


Figure 3. Worst-case execution scenario under SP-CF with fixed-priority scheduling.

Using Lemma 5, DA for SP-CF is derived as follows.

Theorem 2 (DA for SP-CF with fixed-priority scheduling)**.** *Using the maximum execution load of task τ_i in the interval D_k under SP-CF scheduling, the task τ_i can complete its execution within D_k if the following equation is satisfied:*

$$\max_{\tau_j \in \tau_{lp(k)}} C_j + \frac{\sum_{\tau_i \in \tau_{hp(k)}} \min(W_i^{\Phi}(D_k), D_k - C_k + 1)}{m} + C_k \leq D_k \tag{7}$$

Proof. By the proof of Lemma 4, this theorem holds. □

5.2. Da for SP-CF with Edf Scheduling

This subsection derives DA for SP-CF with EDF scheduling. Unlike SP-CF with the fixed-priority scheduling, a job J_i with the later absolute deadline cannot interfere a job of J_k with the earlier absolute deadline according to the scheduling policy of EDF. For example, the third job in Figure 3 cannot participate in $W_i(D_k)$ to interfere J_k because its absolute deadline is later than that of J_k of interest. As shown in Figure 4, the worst-case execution scenario under SP-CF with EDF scheduling occurs when the deadline of the last job of τ_i and the end of the interval of length D_k are aligned.

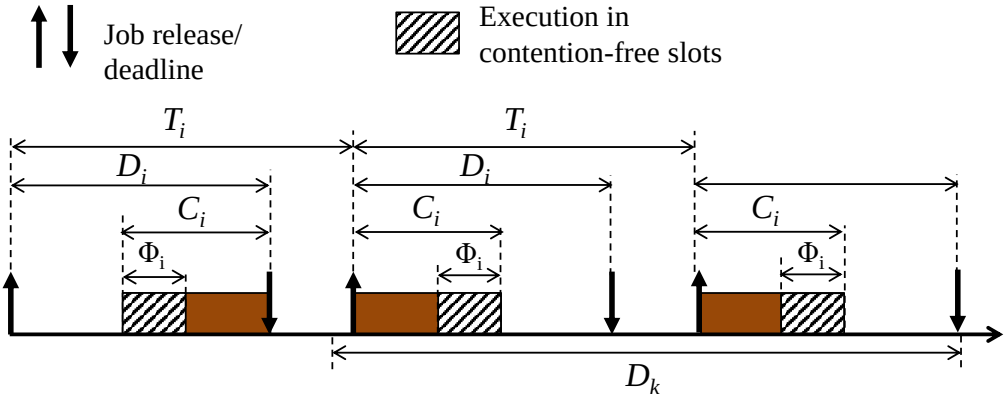


Figure 4. Worst-case execution scenario under SP-CF with EDF scheduling

Using the worst-case execution scenario under SP-CF with EDF scheduling shown in Figure 4, the following lemma derives the execution of task τ_i under SP-CF with EDF scheduling in an interval of length D_k.

Lemma 6 (Execution load of task τ_i under SP-CF with EDF scheduling)**.** *The maximum load $E_i^{\Phi}(D_k)$ of task τ_i within interval D_k under SP-CF with EDF scheduling is given by:*

$$E_i^{\Phi}(D_k) = n_i^{\Phi}(D_k) \cdot (C_i - \Phi_i) + \min(C_i - \Phi_i, D_k - n_i(D_k) \cdot T_i), \tag{8}$$

where $n_i^{\Phi}(D_k)$ is derived by

$$n_i^{\Phi}(D_k) = \left\lfloor \frac{D_k}{T_i} \right\rfloor. \tag{9}$$

Proof. By the proof of Lemma 5, the lemma holds. □

Using Lemma 5, DA for SP-CF with EDF scheduling is derived as follows.

Theorem 3 (DA for SP-CF with EDF scheduling)**.** *Using the maximum execution of task τ_i in the interval D_k under SP-CF with EDF scheduling, the task τ_i can complete its execution within D_k if the following equation is satisfied:*

$$\max_{\tau_j \in \tau \setminus \tau_k} C_j + \frac{\sum_{\tau_i \in \tau_{hp(k)}} \min(E_i^{\Phi}(D_k), D_k - C_k + 1)}{m} + C_k \leq D_k \tag{10}$$

Proof. By the proof of Theorem 2, this theorem holds. □

6. Evaluation

In this section, we assess the efficacy of our proposed SP-CF scheduling methodology, as applied to the fixed-priority and EDF scheduling. We conducted experiments based on a simulator we implemented ourselves in Java SE 21. We consider rate-monotonic (RM) scheduling as the most typical form of fixed-priority scheduling. RM assigns higher priority to tasks with shorter periods. We conduct our evaluation by performing experiments with a custom-built Java simulator, allowing us to ascertain the number of randomly generated task sets that can guarantee the absence of deadline violations under the evaluated techniques.

We evaluate the performance of our schedulability analysis by comparing its results with those of the scheduling process, specifically looking for any deadline violations. This comparison gives us a reliable measure of the scheduler's accuracy, considering the integral relationship between the scheduling algorithm and the real-time analysis technique.

To gauge the efficiency of the methodologies, we create a range of task sets at random. This is achieved through the application of a task set generation method extensively employed in a variety of prior studies [15,27,28]. The task sets generated are of two types, characterized by implicit and constrained deadlines, and are evaluated under four different processor numbers, i.e., $m \in \{2, 4, 8, 16\}$. The utilization (C_i/T_i) of each task is determined by either a bimodal or exponential distribution, with input parameters drawn from the set $\{0.1, 0.3, 0.5, 0.7, 0.9\}$ [11]. Depending on the given bimodal parameter p, a value for C_i/T_i is uniformly selected from the ranges $[0, 0.5)$ and $[0.5, 1)$ with probabilities p and $1 - p$, respectively. For a given exponential parameter $1/\lambda$, the value is derived from the exponential distribution characterized by the probability density function $\lambda \cdot exp(-\lambda \cdot x)$. Each task has a T_i value that is uniformly chosen within $[1, 1000]$, while C_i is determined by the bimodal or exponential parameter, and D_i is randomly selected within the range $[C_i, T_i]$. Based on these parameters, we generate task sets, beginning by creating an initial set of $m + 1$ tasks. The generated task set is then tested against the necessary feasibility condition described in [29]. If the task set does not pass this test, it is discarded and a new set is generated. Conversely, if it meets the criteria, the set is included for further evaluation. This task set is then used as a basis for the next one by adding an additional task, and the process is repeated. The feasibility condition, used during testing, suggests that a task set with a system utilization ($U_{sys} = \sum_{\tau_i \in \mathcal{T}} C_i/T_i$) greater than m cannot be scheduled by

any algorithm. We generate a total of 10,000 task sets for each bimodal or exponential distribution with their respective input parameters (e.g., bimodal distribution with 0.1), each individual value of m (e.g., $m = 2$), and each type of deadline (e.g., implicit deadline). Given the ten different distribution configurations, four different values of m, and two types of deadlines, a total of 400,000 task sets ($10{,}000 \cdot 10 \cdot 4$) are generated.

We demonstrate the applicability of our synthetically produced task sets by illustrating a real-time system along with its task parameters, which is also presented in [30]. A prominent instance of an expansive real-time system is a satellite system. Focusing on this realm, we delve into a reconnaissance satellite system that employs a specialized antenna capable of sending and receiving radio signals to capture images from the target region, regardless of obstructions like nighttime or overcast conditions. The Antenna Control System (ACSW) [31] governs this specific antenna. Within the satellite system, tasks are orchestrated by RM on a specialized RTOS termed RTEMS (Real-Time Executive for Multi-Processor Systems) [32]. The ACSW oversees five main tasks: tHigh, tMilbus, tOne, tTwo, and tSync, with the ensuing functionalities:

- "tHigh" processes each macro command (MCMD) from the ground station queue and forwards relevant data to the other tasks.
- "tMilbus" fetches MCMDs using the MIL-STD-1553B protocol [33] and ensures the authenticity of each MCMD via methods such as CRC prior to queuing.
- "tOne" supervises internal mode shifts, including equipment operations and telemetry relay via the SpaceWire protocol [34].
- "tTwo" manages processes like FDIR, and crafts packets encapsulating system details for transmission to ground control.
- "tSync" prepares operations whenever there's excess computational capability.

Task metrics for ACSW are tabulated in Table 1. Parameters like T_i and D_i are architect-defined, while metrics such as BCET, WCET, and ACET are gauged from actual operational conditions on the target system. This system boasts a 256MB SDRAM and operates on a multi-processor system underpinning the FT Leon3 CPU architecture, clocked at 80 MHz, complemented by SPARC BSP [35], which supports both MIL-STD-1553B (external) and SpaceWire (internal) communication protocols. Notably, tSync operates flexibly, running only when other tasks are not. Task frequencies align with their core purposes: MCMD retrieval every 62.5 ms, MCMD reception every 125 ms, equipment mode switches every 250 ms, and system status updates every 500 ms. The task set crafting technique previously explored yields multiple task combinations with assorted attributes. This versatility makes it adaptable for diverse real-time systems with tasks operating in varying contexts.

Table 1. Task parameters (in milliseconds) for ACSW.

	T_i	D_i	WCET	BCET	ACET
tHigh	62.5	50	2.98	0.08	0.14
tMilbus	125	100	0.54	0.11	0.21
tOne	250	200	30.08	0.05	0.29
tTwo	500	400	231.72	37.7	147.5

We consider the following approaches.

- NPEDF: DA test for non-preemptive EDF that does not allow any preemption. For the test, Equation (4) in Lemma 4 can be applied by substituting $\tau_{lp(k)}$ and $W_i(D_k)$ by $\tau \setminus \tau_k$ and $E_i(D_k)$ where $E_i(D_k) = \left\lfloor \frac{D_k}{T_{i_i}} \right\rfloor \cdot C_i + \min(C_i, D_k - \left\lfloor \frac{D_k}{T_{i_i}} \right\rfloor) \cdot T_i)$.
- SPEDF-CF: DA test for SP-CF with EDF in Theorem 3.
- NPRM: DA test for non-preeemptive RM in Lemma 4 that does not allow any preemption.
- SPRM-CF: DA test for SP-CF with RM in Theorem 3.

To evaluate the efficacy of the various schedulability analysis tests under consideration, we start by determining the quantity of task sets deemed schedulable by each respective technique. This is subsequently compared across different processor counts, ranging from 2 to 16 (denoted as m). We further delve into understanding the effect of the average task set size (represented as $\overline{n}$) and the average utilization of tasks in a set (denoted by $\overline{C_i/T_i}$) on the performance of each schedulability analysis technique. Figure 5 illustrates these considerations when m = 2 and 16. Here, we focus on bimodal distributions with p = 0.9 and exponential distributions with both p = 0.1 and p = 0.9 parameters. Among the ten task utilization distributions, the three are selected due to their distinct $\overline{n}$ (minimal, maximal, and median) and $\overline{C_i/T_i}$ (maximal, minimal, and median). Each subplot in Figure 5 shows the quantity of task sets that each schedulability test deems schedulable against the shifting task set utilization ($U_{sys} = \sum_{\tau_i \in \mathcal{T}} C_i/T_i$) based on the assigned task utilization distribution.

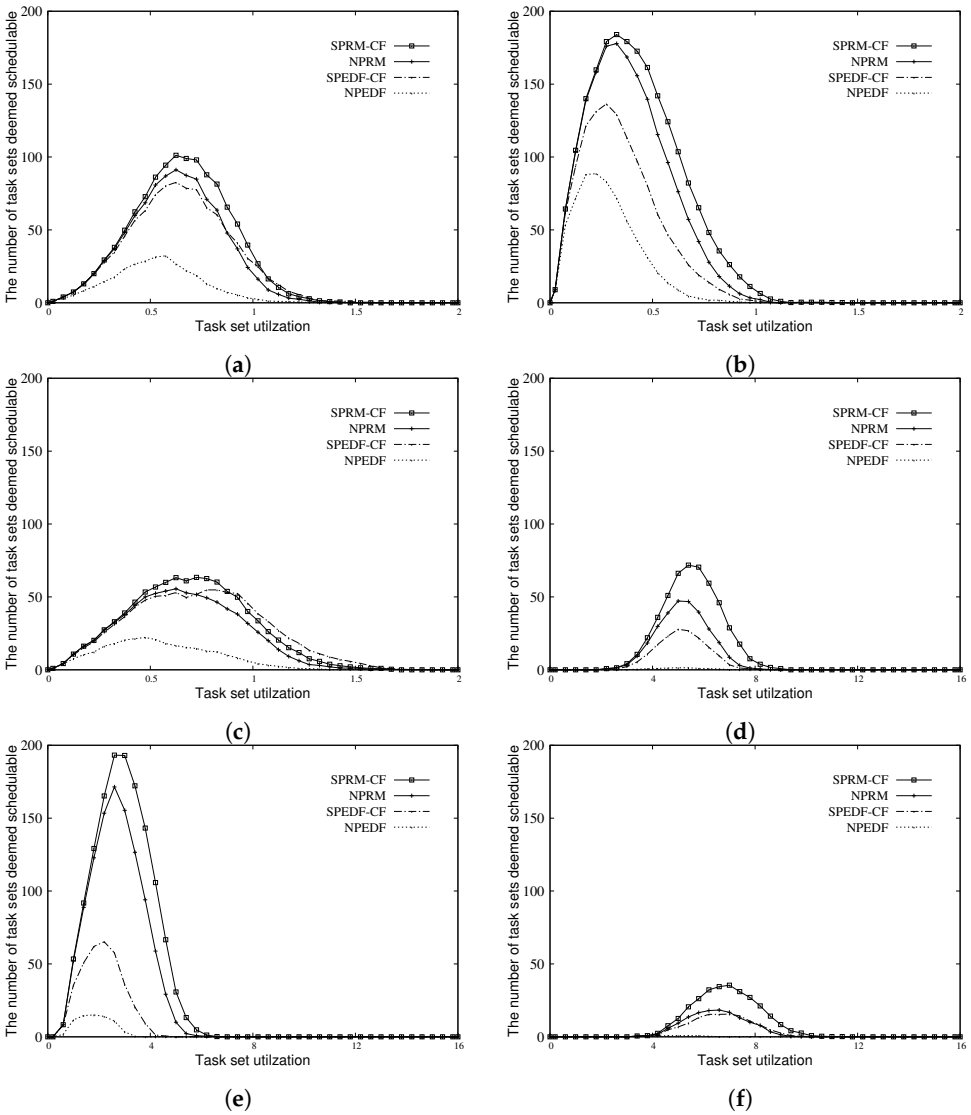


Figure 5. Experiment results for m = 2 and 16. (**a**) Bimodal distribution with 0.9 for m = 2 ($\overline{n}$ = 3.1, $\overline{C_i/T_i}$ = 0.56). (**b**) Exponential distribution with 0.1 for m = 2 ($\overline{n}$ = 11.6, $\overline{C_i/T_i}$ = 0.1). (**c**) Exponential distribution with 0.9 for m = 2 ($\overline{n}$ = 4.3, $\overline{C_i/T_i}$ = 0.34). (**d**) Bimodal distribution with 0.9 for m = 16 ($\overline{n}$ = 19.8, $\overline{C_i/T_i}$ = 0.69). (**e**) Exponential distribution with 0.1 for m = 16 ($\overline{n}$ = 83.2, $\overline{C_i/T_i}$ = 0.1). (**f**) Exponential distribution with 0.9 for m = 16 ($\overline{n}$ = 28.0, $\overline{C_i/T_i}$ = 0.4).

Our initial analysis, grounded on data from Figure 5, leads us to several observations as follows.

O1. As can be seen in Figure 5b,e, all methods show high performance for distributions with a low task utilization (C_i/T_i).
O2. In all cases, the RM series (i.e., LPRM-CF and NPRM) shows higher performance than the EDF series (i.e., LPEDF-CF and NPEDF).
O3. In all cases, the techniques with SP-CF applied dramatically improve the performance of RM and EDF for both $m = 2$ and $m = 16$.

O1 is due to the non-preemptive (e.g., for NPEDF and NPRM) or single-preemptive (e.g., SPEDF-CF and SPRM-CF) characteristics of each approach. In such scheduling, a job of τ_k of interest can be blocked by at most one lower priority job. Therefore, when the distribution has a low average task utilization, the amount of blocking received from lower priority tasks decreases.

O2 arises due to the pessimism in the schedulability analysis of NPEDF and SPEDF-CF, compared to NPRM and SPRM-CF. Unlike RM, in the case of EDF, it assigns priority not to the task itself but to each individual job. Therefore, in the schedulability analysis of NPEDF and SPEDF-CF, when calculating the worst-case response time and comparing it with D_k, it assumes interference from all tasks excluding the τ_k of interest. On the other hand, since NPRM and SPRM-CF only consider tasks with higher priority than the τ_k of interest to calculate the worst-case interference on τ_k, they show better performance than NPEDF and SPEDF-CF in terms of the efficiency of the analysis.

O3 demonstrates the efficiency of SPCF in improving schedulability. As shown in Theorems 2 and 3, under SPCF scheduling, the amount of interference received from higher-priority jobs is reduced to a minimum contention-free maximum of Φ_i. This leads to the improved performance compared to NPRM and NPEDF.

7. Discussion

We discuss the time complexity of the proposed methods as follows. Equation (7) is applied to each task τ_i of the task set τ. For the first term of this equation, $O(n)$ is needed, and for the second term, $O(n)$ is also required. Therefore, the total time complexity is $O(n)$, and when applied to n tasks, it becomes $O(n^2)$. The same goes for Equation (10).

In multi-processor real-time systems, global scheduling allows tasks to run on any processor, providing increased flexibility and aiding load balancing [36]. However, this flexibility comes at the cost of potential migration overheads and cache consistency issues. EDF scheduling method prioritizes jobs with imminent deadlines due to their urgency and is considered optimal for single-processor platforms [1]. Yet, in multi-processor contexts, achieving peak performance entails not just addressing job urgency but also leveraging the parallelism offered by using multiple processors concurrently. When setting priorities at the job level, algorithms like EQDF [37] and SPDF [8] often outshine EDF. We consider global scheduling, and their performance can be further enhanced by integrating them with contention-free policies.

On the flip side, semi-partitioned scheduling mainly designates tasks to specific processors while still allowing some tasks to migrate [38,39]. This approach melds the benefits of task partitioning with the versatility of migration, enhancing cache efficiency, reducing task contention, and curtailing wait times. Nevertheless, migration overheads and algorithmic intricacies remain concerns. For semi-partitioned approaches, maximizing the number of tasks allocated across processors is essential. Once designated, tasks can be managed using ideal single-processor policies: EDF for task-level fixed priority and RM for job-level fixed priority [1]. It is worth noting, however, that task partitioning is a recognized NP-Hard challenge, and foundational partitioning techniques, like bin-packing, can exploit only half of the available multi-processor power to maintain schedulability. The CF policy strives for efficient multi-processor resource utilization. Given that EDF, which optimizes computing resources, is already optimal for single processors, applying the CF policy there would be superfluous. As such, integrating the CF policy within partitioned algorithms delineates a distinct research trajectory.

8. Conclusions

In this paper, we presented SP-CF, a unique single-preemption scheduling strategy incorporating the CF policy. This strategy permits a single preemption during each job's execution, post a priority demotion in accordance with the CF policy. Furthermore, we proposed a novel schedulability analysis methodology, named DA analysis, for SP-CF. This method ensures the punctual execution of each task without deadline violations, applicable not only to fixed-priority scheduling but also to EDF scheduling. Through extensive simulation testing, we showed that SP-CF offers a considerable enhancement in schedulability compared to traditional algorithms like rate-monotonic and earliest deadline first algorithms.

Looking ahead, there are several intriguing avenues for future research. For instance, further improvements to the DA analysis technique can be explored, along with the introduction of novel real-time analysis methodologies such as response-time analysis [40]. It would also be of interest to investigate the applicability of these techniques in mixed-criticality systems, where tasks have differing degrees of importance. Extending the analysis to cyber-physical systems, which involve the intertwined functioning of computational and physical elements, also promises valuable insights.

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

Funding: This work was supported by Incheon National University Research Grant in 2022. This research was also supported in part by the Chung-Ang University Research Grants in 2021.

Data Availability Statement: Not applicable.

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

References

1. Liu, C.; Layland, J. Scheduling Algorithms for Multi-programming in A Hard-Real-Time Environment. *J. ACM* **1973**, *20*, 46–61. [CrossRef]
2. Lee, J.; Chwa, H.S.; Lee, J.; Shin, I. Thread-Level Priority Assignment in Global Multiprocessor Scheduling for DAG Tasks. *J. Syst. Softw.* **2016**, *113*, 246–256. [CrossRef]
3. Lee, J.; Shin, K.G. Development and Use of a New Control Task Model for Cyber-Physical Systems: A Real-Time Scheduling Perspective. *J. Syst. Softw.* **2017**, *126*, 45–56. [CrossRef]
4. Zeng, G.; Matsubara, Y.; Tomiyama, H.; Takada, H. Energy-aware task migration for multiprocessor real-time systems. *Future Gener. Comput. Syst.* **2016**, *56*, 220–228. [CrossRef]
5. Park, S.; Kim, J.H.; Fox, G. Effective real-time scheduling algorithm for cyber physical systems society. *Future Gener. Comput. Syst.* **2016**, *56*, 253–259. [CrossRef]
6. Lee, H.; Lee, J. Limited Non-Preemptive EDF Scheduling for a Real-Time System with Symmetry Multiprocessors. *Symmetry* **2020**, *12*, 172. [CrossRef]
7. Regnier, P.; Lima, G.; Massa, E.; Levin, G.; Brandt, S. RUN: Optimal Multiprocessor Real-Time Scheduling via Reduction to Uniprocessor. In Proceedings of the IEEE Real-Time Systems Symposium (RTSS), Vienna, Austria, 29 November–2 December 2011; pp. 104–115.
8. Chwa, H.S.; Back, H.; Chen, S.; Lee, J.; Easwaran, A.; Shin, I.; Lee, I. Extending Task-level to Job-level Fixed Priority Assignment and Schedulability Analysis Using Pseudo-deadlines. In Proceedings of the IEEE 33rd Real-Time Systems Symposium (RTSS), San Juan, PR, USA, 4–7 December 2012; pp. 51–62.
9. Baker, T.P.; Cirinei, M.; Bertogna, M. EDZL Scheduling Analysis. *Real-Time Syst.* **2008**, *40*, 264–289. [CrossRef]
10. Lee, J.; Easwaran, A.; Shin, I. Maximizing Contention-Free Executions in Multiprocessor Scheduling. In Proceedings of the IEEE Real-Time Technology and Applications Symposium (RTAS), Chicago, IL, USA, 11–14 April 2011; pp. 235–244.
11. Lee, J.; Easwaran, A.; Shin, I. Contention-Free Executions for Real-Time Multiprocessor Scheduling. *ACM Trans. Embed. Comput. Syst.* **2014**, *13*, 1–69. [CrossRef]
12. Cirinei, M.; Baker, T.P. EDZL Scheduling Analysis. In Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS), Pisa, Italy, 4–6 July 2007; pp. 9–18.
13. Davis, R.I.; Burns, A. FPZL Schedulability Analysis. In Proceedings of the RTAS, Chicago, IL, USA, 11–14 April 2011; pp. 245–256.
14. Lee, J.; Shin, K.G. Schedulability Analysis for a Mode Transition in Real-Time Multi-core Systems. In Proceedings of the IEEE Real-Time Systems Symposium (RTSS), Vancouver, BC, Canada, 3–6 December 2013; pp. 11–20.

15. Bertogna, M.; Cirinei, M.; Lipari, G. Schedulability Analysis of Global Scheduling Algorithms on Multiprocessor Platforms. *IEEE Trans. Parallel Distrib. Syst.* **2009**, *20*, 553–566. [CrossRef]
16. Moir, M.; Ramamurthy, S. Pfair Scheduling of Fixed and Migrating Periodic Tasks on Multiple Resources. In Proceedings of the IEEE Real-Time Systems Symposium (RTSS), Phoenix, AZ, USA, 1–3 December 1999.
17. Levin, G.; Funk, S.; Sadowski, C.; Pye, I.; Brandt, S. DP-FAIR: A Simple Model for Understanding Optimal Multiprocessor Scheduling. In Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS), Brussels, Belgium, 6–9 July 2010; pp. 3–13.
18. Baek, H.; Lee, J.; Shin, I. Multi-Level Contention-Free Policy for Real-Time Multiprocessor Scheduling. *J. Syst. Softw.* **2018**, *137*, 36–49. [CrossRef]
19. Ekelin, C. Clairvoyant Non-Preemptive EDF Scheduling. In Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS), Dresden, Germany, 5–7 July 2006; pp. 23–32.
20. Nasri, M.; Kargahi, M. Precautious-RM: A predictable non-preemptive scheduling algorithm for harmonic tasks. *Real-Time Syst.* **2014**, *50*, 548–584. [CrossRef]
21. Nasri, M.; Fohler, G. Non-work-conserving scheduling of non-preemptive hard real-time tasks based on fixed priorities. In Proceedings of the International Conference on Real-Time Networks and Systems, Lille, France, 4–6 November 2015; pp. 309–318.
22. Nasri, M.; Fohler, G. Non-work-conserving non-preemptive scheduling: Motivations, challenges, and potential solutions. In Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS), Toulouse, France, 5–8 July 2016; pp. 165–175.
23. Nasri, M.; Brandenburg, B. Offline Equivalence: A Non-preemptive Scheduling Technique for Resource-Constrained Embedded Real-Time Systems. In Proceedings of the IEEE Real-Time Technology and Applications Symposium (RTAS), Pittsburg, PA, USA, 18–21 April 2017; pp. 75–86.
24. Nasri, M.; Brandenburg, B. An exact and sustainable analysis of non-preemptive scheduling. In Proceedings of the IEEE Real-Time Systems Symposium (RTSS), Paris, France, 5–8 December 2017; pp. 12–23.
25. Nasri, M.; Gerhard, F. Open problems on non-preemptive scheduling of mixed-criticality real-time systems. In Proceedings of the Real-Time Scheduling Open Problems Seminar, Lund, Sweden, 7 July 2015; pp. 17–18.
26. Bertogna, M.; Cirinei, M. Response-Time Analysis for globally scheduled Symmetric Multiprocessor Platforms. In Proceedings of the IEEE Real-Time Systems Symposium (RTSS), Tucson, AZ, USA, 3–6 December 2007; pp. 149–160.
27. Baker, T. An Analysis of EDF Schedulability on a Multiprocessor. *IEEE Trans. Parallel Distrib. Syst.* **2005**, *16*, 760–768. [CrossRef]
28. Andersson, B.; Bletsas, K.; Baruah, S. Scheduling Arbitrary-Deadline Sporadic Task Systems on Multiprocessor. In Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), Kaohisung, Taiwan, 25–27 August 2008; pp. 197–206.
29. Baker, T.P.; Cirinei, M. A Necessary and Sometimes Sufficient Condition for the Feasibility of Sets of Sporadic Hard-deadline Tasks. In Proceedings of the IEEE Real-Time Systems Symposium (RTSS), Rio de Janeiro, Brazil, 5–8 December 2006; pp. 178–190.
30. Baek, H.; Lee, J. Improved schedulability analysis of the contention-free policy for real-time systems. *J. Syst. Softw.* **2019**, *154*, 112–124. [CrossRef]
31. Baek, H.; Lee, H.; Lee, H.; Lee, J.; Kim, S. Improved Schedulability Analysis for Fault-Tolerant Space-Borne SAR System. In Proceedings of the Conference on Korea Institute of Military Science and Technology (KIIT), Deajeon, Republic of Korea, 7–8 June 2018; pp. 1231–1232.
32. RTEMS Community. RTEMS Real-Time Operating System. Available online: https://www.rtems.org (accessed on 14 August 2023).
33. Excalibur Systems. MIL-STD-1553B. Available online: https://www.mil-1553.com (accessed on 14 August 2023).
34. European Space Agency. SpaceWire. Available online: http://spacewire.esa.int (accessed on 14 August 2023).
35. Cobham Gaisler. VxWorks 7 SPARC Architectural Port and BSP. Available online: https://www.gaisler.com (accessed on 14 August 2023).
36. Fisher, N.W. The Multiprocessor Real-Time Scheduling of General Task Systems. Ph.D. Thesis, University of North Carolina, Chapel Hill, NC, USA, 2007.
37. Back, H.; Chwa, H.S.; Shin, I. Schedulability Analysis and Priority Assignment for Global Job-Level Fixed-Priority Multiprocessor Scheduling. In Proceedings of the IEEE Real Time and Embedded Technology and Applications Symposium (RTAS), Beijing, China, 16–19 April 2012; pp. 297–306.
38. Brandenburg, B.B.; Gul, M. Global scheduling not required: Simple, near-optimal multiprocessor real-time scheduling with semi-partitioned reservations. In Proceedings of the IEEE Real-Time Systems Symposium (RTSS), Porto, Portugal, 29 November–2 December 2016; pp. 1–15.
39. Baruah, S.; Fisher, N. The Partitioned Multiprocessor Scheduling of Deadline-Constrained Sporadic Task Systems. *IEEE Trans. Comput.* **2006**, *55*, 918–923. [CrossRef]
40. Joseph, M.; Pandya, P. Finding response times in a real-time system. *Comput. J.* **1986**, *29*, 390–395. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.