TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 01/23 pp1961–1976
DOI: 10.26599/TST.2025.9010031
Volume 31, Number 4, August 2026

Research Article

Research on the Schedulability of Sporadic Tasks in Spaceborne Mixed Real-Time Operating System

Chenghao Song, Lei Qiao*, Mengfei Yang, Maolin Yang, Hongbiao Liu, and Jun Shen

Abstract: A spacecraft is a typical time-sensitive system, and the characteristics of hard real-time and limited resources make the design of real-time scheduling algorithms particularly crucial in spaceborne operating system. With the development of space technology, onboard computer has transformed from a closed system with a single type of task to an open system with mixed sets of tasks. The system allows new tasks to be dynamically loaded during operation, leading to real-time changes in the types and numbers of tasks, further increasing the difficulty of predicting and the uncertainty of the system. Sporadic tasks are more common in actual system and have special temporal characteristics of releasing by a random frequency, making the schedulability determination more complex than that of other types of tasks. Therefore, we focused on the schedulability of sporadic tasks in spaceborne operating system and first classified the situations in which schedulability needs to be determined. For each preemption case in different situations, corresponding determination strategies were proposed based on Response Time Analysis (RTA). Due to RTA's high time complexity, we additionally utilized Interference Bound Function (IBF) to judge the schedulability, thus providing flexible choices for system design. By tracking task's runtime information and analyzing at a finer granularity, our methods reduced pessimism and achieved a better schedulable ratio.

Key words: spaceborne operating system; mixed sets of real-time tasks; sporadic task; schedulability determination

1 Introduction

At present, real-time systems are widely used in various fields, such as autonomous vehicle, industrial robots, digital airplane, phased array radar, even in graphics processing[1–3]. Different from general computing systems, real-time systems have higher requirements for the generation instant of computing results. Some applications, also defined as real-time tasks, running on such systems usually need to meet their own deadline requirements. According to the severity of time constraints, real-time tasks can be divided into hard real-time tasks, soft real-time tasks, and firm real-time tasks. The former does not allow missing deadline, otherwise it may lead to catastrophic

• Chenghao Song, Lei Qiao, and Hongbiao Liu are with Beijing Institute of Control Engineering, Beijing 100190, China. E-mail: hackmyway@163.com; fly2mars@163.com; 2488890051@qq.com.
• Mengfei Yang is with China Academy of Space Technology, Beijing 100094, China. E-mail: yangmf@bice.org.cn.
• Maolin Yang is with School of Information and Software Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China. E-mail: maolyang@uestc.edu.cn.
• Jun Shen is with School of Computing and Information Technology, University of Wollongong, Wollongong 2522, Australia. E-mail: jshen@uow.edu.au.
* To whom correspondence should be addressed.
Manuscript received: 2024-06-28; revised: 2024-08-06; accepted: 2025-03-04

consequences[4]. Due to the special time sensitivity of real-time tasks, some existing traditional scheduling methods, such as Round Robin and FCFS[5], cannot perform well. Therefore, real-time scheduling has become a hot research topic in both academia and industry.

Real-time task scheduling in spacecraft computers is more complex. Spacecraft, as a typical hard real-time system, is strictly limited by volume and heat dissipation constraints, resulting in significant differences in computing power and storage resources compared to ground platforms. The implementation of scheduling strategies should be lightweight, and the computing resources need to be fully utilized as much as possible, which implies that a highly accurate schedulability determination method is crucial. Previously, the type of space tasks in OnBoard Computer (OBC) was relatively single, mainly some safety-critical control tasks such as attitude and orbit control, which are periodically initiated at fixed time instants and therefore also defined as "fixed-point tasks". The targeted scheduling method for such tasks is table-based scheduling strategy[6], which completes runtime scheduling by manually planning the execution timing of tasks in advance before the space vehicle is launched into space. This method trades space for time, which is simple to implement and has strong predictability, but not flexible enough, and can only be applied to situations where the type and numbers of tasks have already been determined.

With the development of onboard electronic technology, the execution of tasks in many subsystems and payloads carried by satellite platforms has been integrated into OBC. Scattered tasks are centralized, and the OBC needs to handle real-time tasks with multiple different attributes. In many aerospace scenarios, such as Mars rovers, manned spacecraft, and satellite constellations[7], the related space missions are extremely complex and different from others, especially in terms of time characteristics, resulting in changes of types of real-time tasks, from a single type to a mixture of multiple types. In addition to fixed-point tasks, there may also be sporadic tasks, such as attitude correction, multi-satellite collaboration, image classification[8], and aperiodic tasks that arrive randomly during system operation like obstacle avoidance. Besides, most spacecraft have evolved from closed systems to open systems, allowing new tasks to be dynamically loaded from external sources during operation, as shown in Fig. 1. Consequently, the ground station is no longer able to figure out all possible situations and is difficult to update spacecraft from a long distance. So spaceborne operating system must be able to analyze the schedulability of mixed sets of tasks online. In actual production, the current approach is still to analyze all possible tasks in advance and manually formulate their execution timing. The handling of emergencies is separated from OBCs, and an independent control system is used, which requires additional microcontrollers and lacks autonomy. Some available researches toward this issue have the problem of insufficient analysis of task types and cannot balance the accuracy of analysis and time complexity well, which is presented in Section 2.

We have previously studied the schedulability of aperiodic tasks in the aforementioned situation. While sporadic tasks are more common in practical systems and have more special timing characteristics which makes it more difficult to determine schedulability. Compared to aperiodic tasks, sporadic tasks trigger multiple instances; compared to fixed-point tasks, the frequency of task's instance is not fixed.

In Section 2, we summarize relevant research and analyze the shortcomings of existing methods when applied in aerospace field. Section 3 introduces the system model, including basic assumptions, task model, and scheduling method, as the basis for the subsequent design of schedulability determination algorithms. Section 4 introduces some critical definitions and theorems with their proof.

In Section 5, we expound detailed illustration and derivation of our strategies. We discover there exists an easy to implement but pessimistic approach to make sufficient schedulability determination, but it can only reach a bad schedulable ratio. So, we sum up all possible preemption cases of sporadic tasks and propose corresponding determination algorithms based

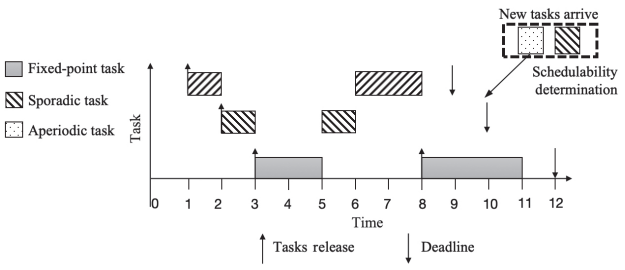


Fig. 1 Mixed sets of tasks in open system.

on Response Time Analysis (RTA) and interference bound function respectively. Section 6 presents simulation experiments and effectiveness of our methods. Finally, a detailed conclusion is drawn.

2 Related Work

In 1973, Liu and Layland[9] proposed the classic Rate-Monotonic (RM) and Earliest Deadline First (EDF) algorithm for periodic tasks in single-core processors and defined the "critical instant" at which the response time of a task reaches maximum. Liu and Layland[9] also provided an upper bound on processor utilization for two algorithms as sufficient condition for schedulability determination. For RM algorithm, when the task set satisfies $\sum_{p=1}^{n} C_p/T_p \leqslant n\left(\sqrt[n]{2}-1\right)$, it is schedulable, where n is the number of tasks, C_p and T_p are the Worst-Case Execution Time (WCET) and the period of the task, respectively. Liu and Layland's research[9] laid an important foundation for the development of real-time scheduling. The "critical instant" provides key idea for determining the schedulability of tasks and has been widely utilized by subsequent research.

The upper bound can only serve as a sufficient condition for schedulability determination, that is, when a task set exceeds this bound, it does not mean the task set cannot be scheduled. Therefore, some sufficient and necessary schedulability determination methods were proposed. Lehoczky et al.[10] proposed a determination strategy for RM algorithm which only needs to check whether there is an instant before the task's deadline that meets the processor requirements of all tasks. However, when the number of tasks is large, it may need to check too many instants, thereby increasing time overhead. Another necessary and sufficient method is RTA[11]. Unlike other methods, RTA can accurately calculate the Worst-Case Response Time (WCRT) of a task, as follows:

$$R_i^{(n+1)} = C_i + \sum_{j=1}^{i} \left\lceil \frac{R_i^{(n)}}{T_j} \right\rceil \cdot C_j$$

When $R_i^{(n+1)} = R_i^{(n)}$, the iteration stops, and $R_i^{(n)}$ is WCRT of the task. Due to the characteristic of iterative computation, when the number of tasks is rational, finding $R_i^{(n)}$ is an NP-hard problem, but RTA is still widely used. The schedulability can be determined by comparing the relative deadline with the WCRT.

The high precision of RTA has obvious advantages when system load is high. While in general, some algorithms that sacrifice certain accuracy but have lower time complexity can be better. Request Bound Function (RBF) is defined as

$$\mathrm{RBF}(\tau_i,\ t) = \left\lceil \frac{t}{T_i} \right\rceil \cdot C_i$$

representing the maximum execution time generated by task τ_i within a time interval of length t[12]. When used for schedulability determination, all tasks are generally considered to release at critical instant, and t can be set as the relative deadline of the task. Obviously, the results obtained by RBF may be greater than t, so there is a certain degree of pessimism. Interference Bound Function (IBF) can also be used to calculate the total execution time of tasks within a certain interval[13]. Compared to RBF, IBF separately calculates the execution time within an incomplete period, thereby improving accuracy. We also utilize IBF to judge the schedulability in Section 5.

As for sporadic tasks, Deadline Monotonic (DM) algorithm, which assigns priority based on task's relative deadline[14], has been proven to be the optimal algorithm for sporadic tasks under fixed-priority scheduling. Liu et al.[15] analyzed the density upper bound of sporadic task sets and provided a sufficient condition for schedulability determination under DM algorithm. Chen et al.[16] constructed a schedulability test framework "k2U", which can generate utilization-based methods for many different types of tasks, including constrained-deadline and arbitrary-deadline sporadic tasks, accordingly having a better generality. Sun et al.[17] proposed a polynomial-time schedulability analysis method for sporadic tasks based on integer programming model, which obtained a significant improvement in the number of deterministic examples compared with existing algorithms.

Initially, real-time tasks are mainly divided into hard real-time tasks and soft real-time tasks based on their tolerance for missing deadlines. The relative scheduling methods mainly include background scheduling, bandwidth reservation, and slack stealing[18–20]. All these methods aim to ensure hard real-time tasks meet their deadlines while minimizing the response time of soft real-time tasks and allowing them to occasionally miss deadline. Many subsequent studies were conducted based on these ideas but with

better real-time performance[21, 22]. While we only consider hard real-time tasks in this paper, and the classification of tasks is mainly based on their arriving patterns. Each task needs to be strictly judged for its schedulability. Compared with more attention paid to reduce response time, our focus is on how to effectively determine reduce schedulability, which is different from the aforementioned researches. How to reduce time overhead and improve the schedulable ratio is the key of schedulability determination. Isovic combined offline scheduling and online scheduling for mixed sets of tasks with complex constraints, solving the schedulability determination problem of three types of tasks: periodic, sporadic, and aperiodic tasks[23]. Compared to background scheduling, this method greatly improves the schedulable ratio of task sets. But But it has high time consumption and is difficult to implement. Coutinho et al.[24] pointed out that when sporadic tasks are jointly scheduled with synchronous periodic tasks, the determination of schedulability only needs to consider the critical instant, while when scheduled with asynchronous periodic tasks, the worst case is unknown. Therefore, they defined the "asynchronous critical instant" of sporadic tasks and then determined the schedulability based on RTA. The "asynchronous critical instant" here only considered two types of tasks, and no corresponding proof process was provided. So we prove this concept and additionally considered aperiodic task, extending to three types of tasks (Section 4).

Regarding mixed scheduling of fixed-point task and other types of tasks, Liu et al.[25] proposed Virtual Zoom Out Period (VZOP), which solved the admission control problem of sporadic tasks. The time complexity of VZOP is only $O(n)$. But it leads to a certain degree of pessimism by using RM's upper bound as the schedulability determination method. When system load increases, the schedulable ratio of task sets decreases rapidly. Based on bandwidth reservation, Liu et al.[26] further proposed method Virtual Utilization Worst Fit (VU-WF) in multi-core processors. A sporadic server with maximum priority was constructed for execution of fixed-point tasks and achieved a higher schedulable ratio by utilizing the determination method in Ref. [10] compared to VZOP. Based on aCoral operating system, Jiang[27] analyzed the WCRT of sporadic task and quantitatively considered the time consumption of admission control under resource constraints. But as the number of tasks increasing, this cost would impose a significant burden on the system. Chen et al.[28] also studied the mixed scheduling when fixed-point and sporadic tasks coexist under shared resources and proposed a lock protocol based on the idea of avoidance blocking which can reduce schedulability loss.

We find there is still few research on mixed scheduling of fixed-point tasks and other types of tasks, especially when aperiodic tasks are additionally considered in open systems. It is also difficult to balance algorithm's time complexity and schedulable ratio for any methods which could directly affect system performance. The above questions are precisely what this paper aims to solve.

3 System Model

System modeling is the foundation for analyzing the schedulability of task sets. As discussed earlier, there are three types of tasks in the system, namely fixed-point task, sporadic task, and aperiodic task, running on a single-core processor with no resource constraints, and the switching cost between different tasks is included in the WCET of each task. We define two states of the system: offline phase and online phase. In offline phase, there are already some fixed-point tasks and sporadic tasks that reflect the basic functions and applications of the system, and the number of these tasks is determined. The tasks are not running, and the system is closed at this state. While during online phase, the system and tasks are dynamically running, and new tasks are allowed to be loaded from external event. The system transforms into an open system. We ideally deem the schedulability determination will be conducted immediately upon arrival of tasks.

The parameters of each task are divided into two categories: Static parameters and Dynamic parameters. The static parameters are determined during the offline phase of system and will not change with the operation of tasks. On the contrary, dynamic parameters require system to record and update in real time to reflect the task's execution status which can contribute to higher precision for schedulability analysis[23, 29].

Fixed-point tasks F_i can be represented by a 4-tuple $F_i = (R_i, C_i, D_i, T_c)$, where all parameters are static. R_i, the arrival offset of F_i; C_i, WCET; D_i, relative deadline; T_c, the system control period. Each fixed-point task releases at a predetermined instant in T_c and completes before the next system control period with the highest priority. Therefore $C_i = D_i$. F_i satisfies

$R_i + C_i \leqslant R_{i+1}$ when $i \neq n$, so any fixed-point task does not interfere with others. Obviously, fixed-point tasks are a kind of special asynchronous periodic task. A task set composed of n fixed-point tasks is represented as $\boldsymbol{T}_F(n) = (F_1, F_2, \ldots, F_n)$.

Sporadic task S_i can be represented by an 8-tuple,

$$S_i = \left(C_i, D_i, T_i, S_i^k, \mathrm{ar}_i^k, d_i^k, E_i^k, f_i^k\right).$$

Static parameters: WCET C_i and relative deadline D_i; T_i, the minimum time between two consecutive invocations of S_i. **Dynamic parameters**: S_i^k, the k-th job of S_i (a single invocation of a task is called a job or an instance); ar_i^k, the arrival time of S_i^k; d_i^k, the absolute deadline of S_i^k; E_i^k, the remaining execution time of S_i^k, with an initial value of C_i; f_i^k, the finish time of S_i^k. The initial value of ar_i^k is -1, indicating that no job of S_i has been triggered yet. The initial value of f_i^k is a maximum value, indicating that the current instance has not finished, i.e., $E_i^k \neq 0$. A task set composed of n sporadic tasks is represented as $\boldsymbol{T}_s(n) = (S_1, S_2, \ldots, S_n)$.

Aperiodic tasks are generally triggered by external events, which do not have periodic characteristics and arrive randomly during system runtime. Therefore, task's parameters are unknown in offline phase. Aperiodic tasks can be represented by a 5-tuple, $A_i = (C_i, \mathrm{ar}_i, E_i, f_i, d_i)$. **Static parameters**: ar_i, the arrival time of A_i; d_i, the absolute deadline; C_i, WCET. **Dynamic parameters**: E_i, the remaining execution time of A_i; f_i, the finish time of A_i. The relative deadline of A_i is the difference between d_i and ar_i. A task set composed of n aperiodic tasks is represented as $\boldsymbol{T}_A(n) = (A_1, A_2, \ldots, A_n)$.

We adopt fixed-priority driven preemptive scheduling algorithm to achieve better predictability and lower difficulty in implementation. Fixed-point tasks are given the highest priority because they often represent some safety-critical control tasks, such as orbit return of manned spacecraft. As for sporadic tasks and aperiodic tasks, their priorities are assigned according to DM algorithm. The priority of a task depends on its relative deadline. A smaller relative deadline means that the task will be allocated a higher priority.

4 Basic Conception

Definition 1 The scheduling window of a sporadic instance S_i^k is $\left[\mathrm{ar}_i^k, d_i^k\right]$, which means that if the execution requirement C_i cannot be met within this interval, S_i^k cannot be scheduled, and therefore S_i cannot be scheduled.

Definition 2 Interference bound from sporadic task $\mathrm{IBF}(S_i, a, b)$ denotes the maximum interference that a sporadic task S_i can cause to lower-priority tasks within a time interval $[a, b]$,

$$\mathrm{IBF}(S_i, a, b) = \left\lfloor \frac{b-a}{T_i} \right\rfloor T_i + \min\{C_i, (b-a) \bmod T_i\}.$$

This formula assumes that S_i requests the processor simultaneously at the critical instant with other tasks. We use it to calculate for arbitrary interval by using task's dynamic parameters.

Lemma 1 When only fixed-point tasks are considered, the response time of S_i^k reaches its maximum when S_i^k releases simultaneously with a fixed-point task F_i. And within scheduling window of S_i^k, the total execution time of all fixed-point tasks is also the largest. In particular, the response time is not necessarily related to execution time.

Proof For each fixed-point task F_i, the releasing time ar_i^k of S_i^k can only locate in $[R_{i-1} + C_{i-1}, R_i + C_i]$. Therefore, the relative positions of ar_i^k to R_i within this interval can be discussed separately.

(1) Response time. In Fig. 2, $\mathrm{ar}_i^k = R_2$, S_i^k releases with F_2 at the same time, when S_i^k releases in advance, i.e., $\mathrm{ar}_i^k < R_2$, because there is no task in $[R_1 + C_1, R_2]$, the response time remains unchanged. However, if the WCET of S_i^k is very small and can be completed within a short idle, then the response time will be respectively small; when $\mathrm{ar}_i^k > R_2$, due to the constant idle time within $[R_2, \text{finish\_time}]$ and the later releasing time, the response time decreases.

(2) Execution time of fixed-point tasks. As shown in Fig. 3, the interval $[\mathrm{ar}_i^k, d_i^k]$ shifts to left and becomes $[x', y']$. Since $[x', \mathrm{ar}_i^k]$ is idle, the total execution time of fixed-point tasks will not decrease unless $[y', d_i^k]$ is also idle; for the interval $[x'', y'']$, on the other hand, the execution time of fixed-point tasks decreases when ar_i^k moves to x'' because $[\mathrm{ar}_i^k, x'']$ is

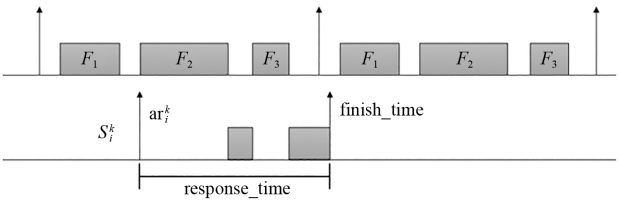


Fig. 2 Response time of S_i^k.

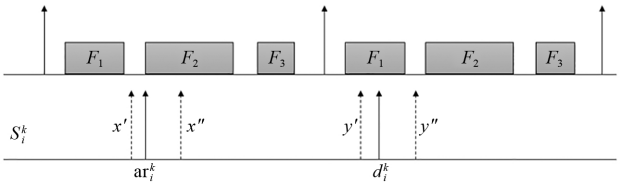


Fig. 3 Execution of fixed-point tasks.

continuously computing. If there is some idle within $[d_i^k, y'']$, the execution time of fixed-point tasks within the overall interval will reduce.

The execution time of fixed-point tasks is not necessarily related to the response time of sporadic instances, as shown in Fig. 4.

Obviously, within the same interval $[\mathrm{ar}_i^k, d_i^k]$, for Case 1, the execution time of fixed-point tasks is greater than Case 2. Assuming the execution requirement of S_i^k can be met during the idle interval in Case 1, the response time in Case 1 will be smaller than that in Case 2.

Lemma 2 Considering fixed-point tasks and sporadic tasks with higher priority (releasing at maximum frequency), when S_i^k and these sporadic tasks release simultaneously with a fixed-point task, the response time of S_i^k reaches its maximum; moreover, the total execution time of all higher-priority tasks within the scheduling window of S_i^k is also maximum.

Proof Lemma 1 shows that if only fixed-point tasks are considered, when a sporadic instance releases simultaneously with a fixed-point task, the response time of this instance and the total execution time of fixed-point tasks within the scheduling window reach maximum. On this basis, sporadic tasks with higher priority are introduced as follows:

(1) Execution time of all tasks with higher priority (including fixed-point tasks and sporadic tasks). In Fig. 5, S_j^k has higher priority than S_i^k. when S_j^k releases in advance, i.e., ar_j^k moves left, the execution time of S_j decreases within $[\mathrm{ar}_i^k, d_i^k]$; because the total idle time

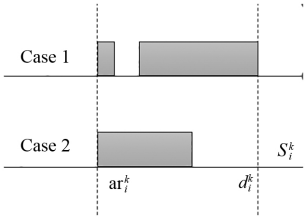


Fig. 4 Relationship between the execution time and the response time.

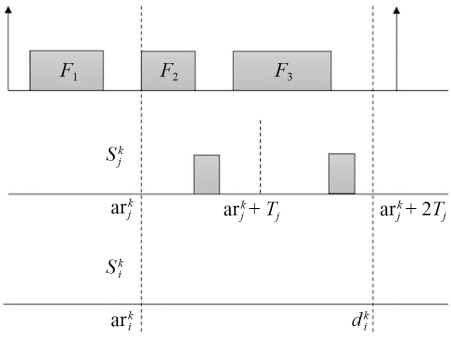


Fig. 5 Execution of fixed-point and sporadic tasks.

for sporadic tasks in $[\mathrm{ar}_i^k, d_i^k]$ is fixed, the available idle time gradually decreases as ar_j^k shifts to right. Both these situations result in a decrease in the execution time of sporadic tasks.

The above process only considers S_i^k releasing simultaneously with F_i. So the cases of releasing at different instants also need to be proved.

Let S_i^k represent a sporadic instance with higher priority now. In Fig. 6, interval $[x, y]$ changes to $[x', y']$, and the corresponding releasing time ar_i^k is also advanced to x'. According to Theorem 1, the execution time of fixed-point tasks in $[x', y']$ remains unchanged or decreases. Since both $[x', x]$ and $[y', y]$ are idle, the execution time of fixed-point tasks in the new interval remains unchanged. Assume the length of $[x', y']$ is twice as the period of S_i^k. S_i triggers at the maximum frequency. In Case 2, ar_i^k continues to move left by a certain value Δ. Since there is no idle in Δ, the execution time of S_i in $[x', y']$ does not reduce. At the same time, due to the advance of $\mathrm{ar}_i^k + 2T_i$, a new period of S_i enters, resulting in an increment of execution time of S_i in $[x', y']$, as indicated by the arrow.

In fact, the increment of execution time needs to meet the following requirements: the idle time in the

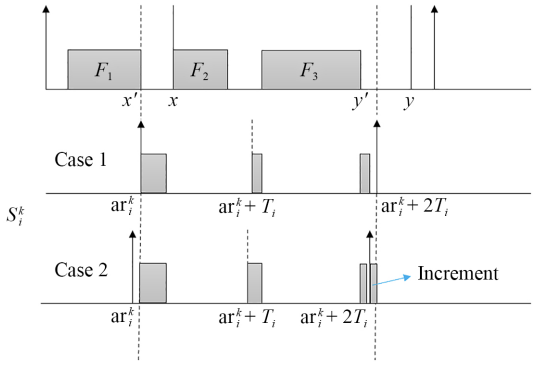


Fig. 6 Releasing at different instants with fixed-point task.

left interval is less than that in the rigth interval after moving. In Case 2 of Fig. 6, it is shown that the idle time in $[\mathrm{ar}_i^k, x']$ is less than idle time in $[\mathrm{ar}_i^k + 2T_i, y']$. This is because during the left shift of ar_i^k, the idle time in the left interval leads to a decrease in the total execution time of sporadic tasks, while the idle time in the rigth interval allows new sporadic instance to be executed. The total change in execution time of sporadic tasks depends on the relationship between the idle time in the left and right intervals and the WCET of sporadic tasks.

We can increase the execution time of fixed-point tasks by moving the interval to left. In Fig. 6, moving x' to ar_i^k in Case 2 changes the analysis interval to $[\mathrm{ar}_i^k, \mathrm{ar}_i^k + 2T_i]$. The execution time of S_i in this new interval stays unchanged, and the execution time of fixed-point tasks increases. So the total execution time still increases compared to the initial analysis interval $[x', y']$. According to this rule, we can continuously move ar_i^k and adjust the analysis interval until its left point is equal to the offset of F_1. Now sporadic tasks and fixed-point tasks release simultaneously again.

(2) Response time. When a sporadic task with higher priority releases simultaneously with a fixed-point task, regardless of whether this sporadic task releases earlier or later, it may cause sporadic task with lower priority to be executed earlier, thereby reducing its response time.

When a task with lower priority, represented by S_r, does not release at the same time with fixed-point task, analysis can still be carried out using Fig. 6. There are two situations that may increase the response time, as follows:

(i) Assuming the response time of S_r in Case 1 is equal or very close to $\mathrm{ar}_i^k + T_i$, $[\mathrm{ar}_i^k, \mathrm{ar}_i^k + 2T_i]$ shifts left by Δ. Since there is no idle in Δ, interference generated by sporadic task with higher priority in $[\mathrm{ar}_i^k, \mathrm{ar}_i^k + T_i]$ does not decrease. At the same time, $\mathrm{ar}_i^k + T_i$ advances, causing the execution of S_r to be preempted, resulting in an increase in the response time.

(ii) $[\mathrm{ar}_i^k, \mathrm{ar}_i^k + 2T_i]$ shifting to left increases the execution time of all instances of S_i within $[x', y']$. If the response time of S_r in Case 1 is equal or very close to $\mathrm{ar}_i^k + 2T_i$, the newly added execution time of S_i will increase the response time of S_r. Due to the execution of fixed-point tasks within Δ, if the releasing time of S_r moves from x' to ar_i^k, its response time will further increase. Similarly, according to this rule, the releasing time of S_i and S_r can continuously shift to left until it is equal to the arrival time of F_1. If the releasing time continuously moves the idle time before F_1 will cause S_i to be executed earlier, thereby reducing its response time.

5 Schedulability Determination of Sporadic Tasks

Under mixed scheduling of different types of tasks, there is no such a critical instant likes RM that can directly determine the schedulability. A simple but pessimistic method is to assume that all tasks release at the same time (since each fixed-point task has a fixed offset, this situation will never occur). The response time of sporadic instance with lower priority reaches a considerable extreme value. We call this method Over-Pessimistic Method (OPM), and our method shows that the accuracy of analysis can be significantly improved by utilizing task's dynamic parameters.

In our previous work[30], we provided a method for calculating the execution time of fixed-point tasks within any interval $\mathrm{RT}_F(a, b)$, where a and b are the left and right endpoints of the interval. In addition, we also defined $\mathrm{IS}(S_i, R_{A_i}^{(n)})$, where $R_{A_i}^{(n)}$ is the result of the n-th iteration in RTA method. $\mathrm{IS}(S_i, R_{A_i}^{(n)})$ utilizes the runtime information of sporadic tasks and can calculate the total execution time generated by sporadic tasks with higher priority in each iteration of RTA. On this basis, we solved the admission control problem of aperiodic tasks and got the worst-case response time of aperiodic tasks. The above results are used in the following analysis.

There are three situations where the schedulability of sporadic tasks needs to be determined, as follows:

(1) Schedulability analysis on initial sporadic tasks when system is not running, i.e., offline phase.

(2) During online phase of system, new sporadic task may dynamically request to join the system. Therefore, it is necessary to determine the schedulability of this new task and other tasks with lower priority.

(3) Aperiodic tasks with higher priority that randomly arrive during runtime may preempt existing sporadic tasks, so their schedulability need to be tested again.

5.1 Schedulability determination in offline phase

According to Lemma 2, when a sporadic task with lower priority and all other sporadic tasks with higher

priority release simultaneously at a fixed-point task's offset, the response time of this sporadic task reaches its maximum. Therefore, we can traverse each fixed-point task and find out the worst case. The schedulability can be determined by comparing the relationship between the maximum response time and the relative deadline of the task.

Assuming the sporadic task to be determined is S_j, one of the sporadic tasks with higher priority is S_j, and the set of sporadic tasks $\boldsymbol{T}_s(n) = (S_1, S_2, \ldots, S_n)$ is arranged by descending order according to task's priority, therefore $j \leqslant i$. The execution time generated by S_j during time interval t is recorded as $\mathrm{ReT}_j(t)$ and $\mathrm{ReT}_j(t) = \left\lceil t/T_j \right\rceil \cdot C_j$, when S_j releases with S_i at the same time. Therefore, the execution time generated by all sporadic tasks with higher priority during t is $\sum_{j=1}^{i-1} \mathrm{ReT}_j(t)$; the execution time of fixed-point tasks is $\mathrm{RT}_F(R_m, R_m + t)$, where R_m is the offset of the m-th fixed-point task.

On this basis, the WCRT of S_i can be calculated via RTA. The initial iterative value is equal to the execution time of S_i, i.e., C_i. The iterative calculation is shown in Algorithm 1.

5.2 Schedulability determination when new sporadic tasks arrive

Since the instants when new sporadic tasks require to join the system are random, the determination of the schedulability needs to consider whether the current instance is preempted by aperiodic tasks. We traverse all aperiodic tasks, and the interference caused by these tasks on sporadic instance is the sum of their remaining execution time.

Let $\boldsymbol{T}_{A\_h}(n)$ denotes the set of aperiodic tasks with higher priority and n is the number of tasks. The interference generated by these tasks can be recorded as $\sum_{A_i \in \boldsymbol{T}_{A-h}(n)} E_i$. The newly arriving sporadic task to be determined is S_i, the arrival time of its first instance S_i^1 is ar_i^1, that is, the current system time when S_i requests to join. Based on whether $\sum_{A_i \in \boldsymbol{T}_{A-h}(n)} E_i$ is equal to 0, there are following discussions:

(1) $\sum_{\boldsymbol{A_i} \in \boldsymbol{T}_{A-h}(n)} \boldsymbol{E_i} \neq \boldsymbol{0}$**.** The execution time of aperiodic tasks in $\boldsymbol{T}_{A\_h}(n)$ only preempts the current sporadic instance and does not affect the next instance. This is because if there is a preemption for the next sporadic instance, it indicates that the aperiodic tasks have not been completed within the scheduling window of S_i^1, and S_i^1 will definitely not be executed, thus missing its deadline. Within the scheduling window $[\mathrm{ar}_1^1, d_1^1]$ of S_i^1, the execution time of fixed-point tasks can be calculated through RT_F; the interference generated by sporadic tasks with higher priority can also be obtained

Algorithm 1 RTA method for determining the schedulability of sporadic tasks in offline stage

Input: $\boldsymbol{T}_F(n) = (F_1, F_2, \ldots, F_n)$, $\boldsymbol{T}_s(n) = (S_1, S_2, \ldots, S_n)$

Output: WCRT of S_i

```
1: int worst_response_time = 0
2: int current_iteration = C_i
3: int last_iteration = 0
4: for R_m in T_F(n)
5: while current_iteration != last_iteration
6:     last_iteration = current_iteration;
7:     current_iteration = RT_F(R_m, R_m + last_iteration) +
       sum_{j=1}^{i-1} ReT_j(last_iteration) + C_i;
8:     end while
9:     if current_iteration > worst_response_time
10:    worst_response_time = current_iteration;
11: return worst_response_time
```

Note: Only when worst_response_time is less than or equal to D_i, S_i can be deemed as scheduled.

Algorithm 2 RTA method for determining the schedulability of current sporadic instance in online stage

Input: $\boldsymbol{T}_F(n) = (F_1, F_2, \ldots, F_n)$, $\boldsymbol{T}_s(n) = (S_1, S_2, \ldots, S_n)$, $\boldsymbol{T}_A(n) = (A_1, A_2, \ldots, A_n)$

Output: WCRT of S_i^1

```
1: int worst_response_time = 0
2: int current_iteration = C_i
3: int last_iteration = 0
4:    while current_iteration != last_iteration
5:       last_iteration = current_iteration;
6:       current_iteration = sum_{j=1}^{i} IS(S_j, last_iterator) + C_i +
         RT_F(ar_i^1, ar_i^1 + last_iteration) + sum_{A_i ∈ T_{A-h}(n)} E_i;
7: end while
8: return worst_response_time
```

Note: It is not only necessary to determine the schedulability of current sporadic instance, but also to analyze whether S_i can meet its deadline in worst case (as it is difficult to determine whether the current instance has the maximum response time among all instances, even it is preempted by aperiodic tasks). The determination method is the same as Section 5.1.

by using tasks' dynamic parameters. At this point, the schedulability of S_i^1 can be determined, shown in Algorithm 2.

(2) $\sum_{A_i \in \boldsymbol{T}_{A-h}(n)} \boldsymbol{E_i} = \boldsymbol{0}$**.** It indicates that tasks in $\boldsymbol{T}_{A_h}(n)$ do not **directly** preempt S_i^1, but this does not mean S_i^1 is not be disturbed by aperiodic tasks. Consider the following example:

At the moment ar_i^1, aperiodic tasks have been completed, $\sum_{A_i \in \boldsymbol{T}_{A-h}(n)} E_i = 0$. The priority of S_{high}^k is greater than S_i but less than aperiodic tasks, so although S_{high}^k releases at $\mathrm{ar}_{\mathrm{high}}^k$, it is not executed until F_1 finishes. In this case, aperiodic tasks with higher priority **indirectly** affect the execution of S_i, possibly making the worst case in Lemma 2 no longer hold. As shown in Fig. 7, if $\mathrm{ar}_{\mathrm{high}}^k$ is equal to ar_i^1, assuming the response time at this time reaches the maximum in Lemma 2, it is obvious that if $\mathrm{ar}_{\mathrm{high}}^k$ shifts to left, the response time of S_i may increase or remain unchanged. Therefore, it cannot be simply confirmed the aperiodic tasks will not interfere with S_i^1 if $\sum_{A_i \in \boldsymbol{T}_{A-h}(n)} E_i = 0$, and itis still necessary to analyze its schedulability of currentinstance and worst case.

RTA can provide a sufficient and necessary test method, but the time complexity is considerable. Judgment for new sporadic tasks occurs in online phase, and unlike the offline stage, the time cost of algorithm should not be too high. Although the RTA method can accurately determine the worst-case response time of a task, it also brings the problem of heavy computing overhead. IBF can also be used to determine the schedulability of sporadic tasks by utilizing Algorithms 3 and 4. IBF reduces time complexity, but introduces a certain degree of pessimism, which may lead to a decrease in the schedulable ratio of task sets. In actual system design,

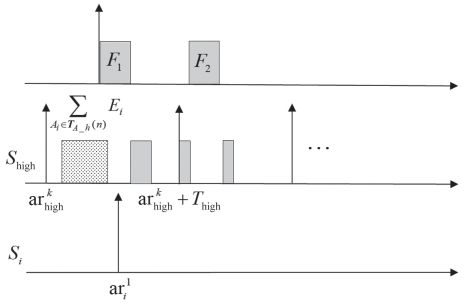


Fig. 7 Indirectly effect of aperiodic tasks on sporadic tasks.

Algorithm 3 IBF method for determining the schedulability of current instance in online stage

Input: $\boldsymbol{T}_F(n) = (F_1, F_2, \ldots, F_n)$, $\boldsymbol{T}_s(n) = (S_1, S_2, \ldots, S_n)$ $\boldsymbol{T}_A(n) = (A_1, A_2, \ldots, A_n)$
Output: schedulability of sporadic instance

bool schedulable = true
int interference_time_s = 0
int interference_time_total = 0
for j from 1 to i–1
  if $\mathrm{ar}_j^k = -1$
    interference_time_s += IBF(S_j, ar_i^1, d_i^1);
  else if $\mathrm{ar}_j^k \neq -1$
  if $\mathrm{ar}_j^k + T_j \leqslant \mathrm{ar}_i^1$
    interference_time_s += IBF(S_j, ar_i^1, d_i^1);
  if $f_j^k \leqslant \mathrm{ar}_i^1 \leqslant \mathrm{ar}_i^k + T_i$
    interference_time_s += IBF(S_j, $\mathrm{ar}_j^k + T_i$, d_i^1);
  if $\mathrm{ar}_i^1 \leqslant f_j^k$
    interference_time_s += E_j^k + IBF(S_j, $\mathrm{ar}_j^k + T_i$, d_i^1);
    interference_time_total = interference_time_s + $\sum_{A_i \in \boldsymbol{T}_{A-h}(n)} E_i$ + $\mathrm{RT}_F(\mathrm{ar}_i^1, d_i^1)$;
if interference_time_total + $C_i > D_i$;
  schedulable = false;
return schedulable

Note: The time complexity of Algorithm 3 is $O(\max(n, m, p))$, where n, m, and p represent the numbers of fixed-point tasks, sporadic tasks, and aperiodic tasks with higher priority, respectively. max(·) takes the maximum value.

Algorithm 4 IBF method for determining schedulability in worst case

Input: $\boldsymbol{T}_F(n) = (F_1, F_2, \ldots, F_n)$, $\boldsymbol{T}_s(n) = (S_1, S_2, \ldots, S_n)$
Output: schedulability of sporadic instance

bool schedulable = true
int interference_time_total = 0
int interference_time_max = 0
for R_m in $\boldsymbol{T}_F(n)$
  interference_time_total = $\sum_{j=1}^{i-1} \mathrm{IBF}(S_j, R_m, R_m + D_i)$ + $\mathrm{RT}_F(R_m, R_m + D_i)$;
  if interference_time_total > interference_time_max
  interference_time_max = interference_time_total;
if interference_time_max + $C_i > D_i$
  schedulable = false;
return schedulable

Note: The time complexity is *O(nm)*, where n and m are the numbers of fixed-point tasks and sporadic tasks with higher priority.

IBF or RTA can be dynamically selected based on system requirements and parameters of tasks. The main difference for IBF is the calculation of execution time of sporadic tasks. The pseudocode is shown in Algorithm 3.

Due to the arrival of new tasks, sporadic tasks with lower priority may not be schedulable. So it is necessary to re-evaluate the schedulability of all these tasks. Not only does it need to determine the schedulability in the worst case (Lemma 2) but also need to analyze whether the current instance will miss its deadline. Assuming at the arrival time ar_i^1 of the new sporadic task S_i, the current in stance of sporadic task S_{low} with lower priority is S_{low}^k, and the sum of the remaining execution time of aperiodic tasks with higher priority than S_{low} is E_A. There are following discussions:

(1) $\mathbf{ar}_{\mathrm{low}}^{\boldsymbol{k}} < \mathbf{ar}_{\boldsymbol{i}}^{\mathbf{1}} < \mathbf{ar}_{\mathrm{low}}^{\boldsymbol{k}} + \boldsymbol{T}_{\mathrm{low}}$**.** In Fig. 8, at ar_i^1, if $E_{\mathrm{low}}^k \neq 0$, it is necessary to determine the schedulability of current instance. The new scheduling window can be selected as $[\mathrm{ar}_i^1, d_{\mathrm{low}}^k]$ to analyze whether the remaining execution time could be satisfied. The determination method is similar to the aforementioned process, and IBF or RTA can also be used separately.

When $E_{\mathrm{low}}^k = 0$, if there are no aperiodic tasks or if the finish time of aperiodic task f_A is less than f_{low}^k of S_{low}^k, only the worst case in Lemma 2 needs to be considered; otherwise, regardless of whether E_A is equal to 0 or not, referring to the previous analysis, aperiodic tasks may directly or indirectly interfere with the next instance S_{low}^{k+1}. Due to the possibility of the large number of sporadic tasks to be redetermined, a relatively pessimistic judgment method can be adopted to control the time consuming: directly consider the worst case in Lemma 2, where the response time is the largest. Firstly, calculate the sum of execution time of fixed-point tasks and all sporadic tasks with higher priority in an interval of length D_{low} by using IBF. On this basis, add the sum of execution time of aperiodic tasks with higher priority to obtain a relatively pessimistic total execution time of all tasks. By comparing this value with D_{low}, schedulability can be determined.

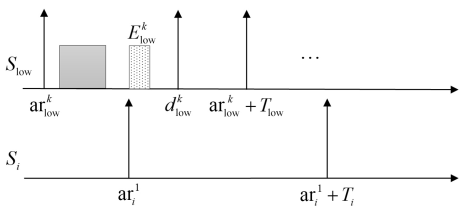


Fig. 8 Preemption of new task on sporadic tasks.

(2) $\mathbf{ar}_{\mathrm{low}}^{\boldsymbol{k}} + \boldsymbol{T}_{\mathrm{low}} \leqslant \mathbf{ar}_{\boldsymbol{i}}^{\mathbf{1}}$**.** In Fig. 9, if at ar_i^1, $E_{\mathrm{low}}^k = 0$ and $\mathrm{ar}_i^1 \geqslant \mathrm{ar}_{\mathrm{low}}^k + T_{\mathrm{low}}$, regardless of whether E_A is equal to 0 , the next instance S_{low}^{k+1} may be affected by aperiodic tasks. Obviously, the worst case occurs when the releasing time of S_{low}^{k+1} is equal to ar_i^1 and all tasks trigger at maximum frequency. At this point, the analysis interval can be selected as $[\mathrm{ar}_i^1, d_{\mathrm{low}}^{k+1}]$, and the execution requirement of S_{low}^{k+1} is its execution time C_{low}. Similarly, when there are currently no aperiodic tasks or when f_A is less than f_{low}^k, only the worst case in Lemma 2 needs to be considered.

(3) $\mathbf{ar}_{\mathrm{low}}^{\boldsymbol{k}} = \mathbf{-1}$**.** It indicates that no instance of S_{low} has been triggered yet. Like discussion in (2), the worst case occurs when $\mathrm{ar}_{\mathrm{low}}^1$ equals to ar_i^1.

In fact, the new sporadic task may also affect the schedulability of some aperiodic tasks. Our previous work[30] can effectively solve this problem.

5.3 Schedulability determination when aperiodic tasks arrive

When aperiodic tasks arrive, the schedulability of sporadic tasks with lower priority needs to be re-evaluate. We assume that the worst-case response time of aperiodic task A_i is $\mathrm{Response}_i$. Sporadic tasks with lower priority is S_j, with the last instance S_j^k upon A_i arriving. There are following situations between S_j^k and A_i:

(1) $\boldsymbol{A_i}$ **does not directly preempt** $\boldsymbol{S_j^k}$**.** If $\mathrm{ar}_i \geqslant \mathrm{ar}_j^k$, $\mathrm{Response}_i \leqslant S_j^k + T_j$ and the remaining execution time E_j^k of S_j^k is equal to 0, A_i will not directly cause any interference. However, similar to the situation shown in Fig. 7, it cannot be determined whether A_i indirectly affects the next sporadic instance S_j^{k+1}. Therefore, it is necessary to determine the schedulability of S_j^{k+1}, and the relatively pessimistic method described in Section 5.2 can be used.

(2) $\boldsymbol{A_i}$ **preempts current sporadic instance.** In

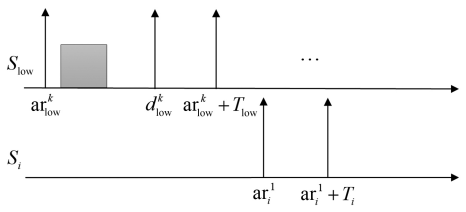


Fig. 9 Case where the next sporadic instance has not yet arrived.

Fig. 10, obviously, at ar_i, the remaining execution time of S_j^k is not zero (if $\mathrm{Response}_i \geqslant d_j^k$, S_j^k will inevitably miss its deadline). Since the worst-case schedulability judgment has been passed when S_j joined the system, it is only necessary to analyze whether the remaining execution time of the current instance can be satisfied before its deadline. The analysis method can still use IBF (Algorithm 5) and RTA (Algorithm 6) respectively, with interval selected as $[\mathrm{ar}_i, d_j^k]$. Let E_{total} represent the sum of remaining execution time of aperiodic tasks with higher priority than S_j. The pseudocodes for the two methods are shown in Algorithms 5 and 6.

(3) A_i influences next sporadic instance. As shown in Fig. 11, at ar_i, when $E_j^k = 0$ and $\mathrm{Response}_i > \mathrm{ar}_j^k + T_j$, A_i will cause interference to the $(k+1)$-th instance S_j^{k+1} of S_j. For S_j, T_j is the minimum arrival interval between two consecutive instances. In actual system operation, the arrival interval of sporadic instances may

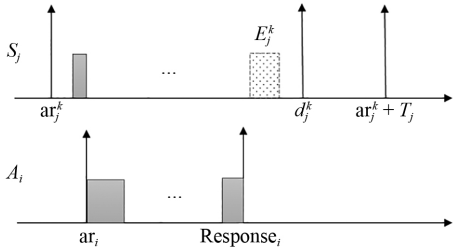


Fig. 10 Directly preemption of aperiodic tasks on sporadic task instance.

Algorithm 5 RTA method for determining the schedulability of current instance when aperiodic tasks arrive

Input: $\boldsymbol{T}_F(n) = (F_1, F_2, \ldots, F_n)$, $\boldsymbol{T}_s(n) = (S_1, S_2, \ldots, S_n)$, $\boldsymbol{T}_A(n) = (A_1, A_2, \ldots, A_n)$
Output: schedulability of sporadic instance

1: **int** worst_response_time = 0
2: **bool** schedulable = true
3: **int** current_iteration = E_j^k
4: **int** last_iteration = 0
5: **while** current_iteration != last_iteration
6: last_iteration = current_iteration;
7: current_iteration = $E_{\mathrm{total}} + E_j^k + \mathrm{RT}_F(\mathrm{ar}_i, \mathrm{ar}_i + \mathrm{last\_iteration}) \sum_{m=1}^{j} \mathrm{IS}(S_m, \mathrm{last\_iterator})$;
8: worst_response_time = current_iteration;
9: **if** current_iteration > $(d_j^k - \mathrm{ar}_i)$
10: schedulable = false;
11: **return** schedulable

Algorithm 6 When aperiodic tasks arrive, IBF method for determining the schedulability of current instance

Input: $\boldsymbol{T}_F(n) = (F_1, F_2, \ldots, F_n)$, $\boldsymbol{T}_s(n) = (S_1, S_2, \ldots, S_n)$, $\boldsymbol{T}_A(n) = (A_1, A_2, \ldots, A_n)$
Output: schedulability of sporadic instance

1: **bool** schedulable = true
2: **int** interference_time_s = 0
3: **int** interference_time_total = 0
4: **for** m from 1 to $j-1$
5: **if** $\mathrm{ar}_m^k = -1$
6: interference_time_s += $\mathrm{IBF}(S_m, \mathrm{ar}_i, d_j^k)$;
7: **else if** $\mathrm{ar}_m^k \neq -1$
8: **if** $\mathrm{ar}_m^k + T_m \leqslant \mathrm{ar}_i$
9: interference_time_s += $\mathrm{IBF}(S_m, \mathrm{ar}_i, d_j^k)$;
10: **if** $f_m^k \leqslant \mathrm{ar}_i \leqslant \mathrm{ar}_m^k + T_m$
11: interference_time_s += $\mathrm{IBF}(S_m, \mathrm{ar}_m^k + T_m, d_j^k)$;
12: **if** $\mathrm{ar}_i \leqslant f_m^k$
13: interference_time_s += $E_j^k + \mathrm{IBF}(S_m, \mathrm{ar}_m^k + T_m, d_j^k)$;
14: interference_time_total = interference_time_s + $E_{\mathrm{total}} + \mathrm{RT}_F(\mathrm{ar}_i, d_j^k)$;
15: **if** interference_time_total + $E_j^k > (d_j^k - \mathrm{ar}_i)$
16: schedulable = false;
17: **return** schedulable

Note: The time complexity of Algorithm 6 is $O(\max(n, m, p))$, the same as that of Algorithm 3.

be greater than T_j. In fact, when S_j^{k+1} releases as early as possible, its response time reaches the worst case. S_j^{k+1} can only be executed after $\mathrm{Response}_i$. As shown in Fig. 11. when $\mathrm{ar}_j^{k+1} < \mathrm{Response}_i$, the earlier S_j^{k+1} releases, the greater the response time will be. Therefore, when the value of ar_j^{k+1} is $\max(\mathrm{ar}_i, \mathrm{ar}_j^k + T_j)$, the response time of S_j^{k+1} reaches its maximum. The reason for the value of ar_j^{k+1} is that ar_i can be either greater than or equal to $\mathrm{ar}_j^k + T_j$, or less than $\mathrm{ar}_j^k + T_j$. When $\mathrm{ar}_i \geqslant \mathrm{ar}_j^k + T_j$, the analysis interval is $[\mathrm{ar}_i, d_j^{k+1}]$,

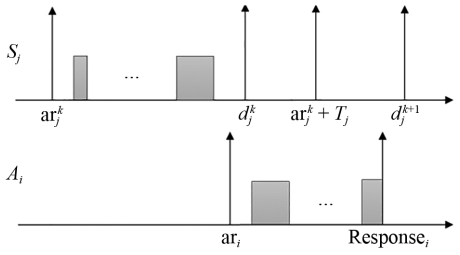


Fig. 11 Preemption of aperiodic tasks on the next sporadic instance.

and regardless of the method used, IBF or RTA, it is similar to previous analysis processes. When $\mathrm{ar}_i < \mathrm{ar}_j^k + T_j$, the value of ar_j^{k+1} is $\mathrm{ar}_j^k + T_j$, two judgment ideas are given below.

(1) In interval $[\mathrm{ar}_i, d_j^{k+1}]$, directly use IBF to find out the total execution time requirement of all tasks in the worst-case and compare this value with the interval length. IBF itself is a relatively pessimistic method. Therefore, when the total execution time is less than or equal to $d_j^{k+1} - \mathrm{ar}_i$, it indicates that the actual execution time must also be less than or equal to the interval length. That is, there exists an idle interval inside $[\mathrm{ar}_i, d_j^{k+1}]$, and because $[\mathrm{ar}_i, \mathrm{Response}_i]$ is constantly computing, the idle part can only be in $[\mathrm{Response}_i, d_j^{k+1}]$. Therefore, the execution demand of sporadic instance can be met before its deadline. When the total execution requirement is greater than $d_j^{k+1} - \mathrm{ar}_i$, S_j^{k+1} is directly judged as unschedulable.

(2) By dynamically recording the runtime information of tasks, we can obtain the dynamic parameters of sporadic instances and aperiodic tasks and use this information to assist schedulability determination. However, the system generally cannot possess such information at a certain instant in the future, unless some constraints are added.

Some assumptions are established in Fig. 12: ① sporadic tasks within $[a, b]$ are triggered at maximum frequency; ② before instant b, all instances generated by tasks with higher priority have been completed. It can be inferred that under the premise of above assumptions, the time relationship between b and the sporadic instance is as follows: if b locates in the scheduling window of S_j^m, then

$$m = k + 1 + \left\lfloor \frac{b - a - (\mathrm{ar}_j^k + T_j - a)}{T_j} \right\rfloor = k + 1 + \left\lfloor \frac{b - \mathrm{ar}_j^k - T_j}{T_j} \right\rfloor.$$

Therefore, among all instances of S_j with a releasing time greater than or equal to b, the minimum arrival time is ar_j^{m+1}. When $a \geqslant \mathrm{ar}_j^k + T_j$, $m = k + \lceil (b-a)/T_j \rceil$.

For sporadic instances in Fig. 11, the analysis

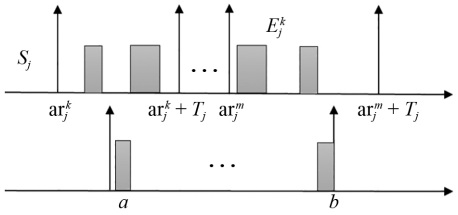


Fig. 12 Prediction of sporadic task.

interval can be selected as $[\mathrm{Response}_i, d_j^k]$. Assuming the set of aperiodic tasks with priority lower than A_i but higher than S_j is $T_{\mathrm{mid}}(n)$, and their remaining execution time is $\sum_{A_q \in \boldsymbol{T}_{\mathrm{mid}}(n)} E_q$, So the interference of these aperiodic tasks to S_j at $\mathrm{Response}_i$ is still $\sum_{A_q \in \boldsymbol{T}_{\mathrm{mid}}(n)} E_q$ (these tasks have not been executed before $\mathrm{Response}_i$).

For the interference from sporadic tasks with higher priority, all sporadic tasks with priority greater than S_j can be divided into two parts: one part has the priority greater than A_i, denoted as S_h, and the other part has a priority greater than S_j but less than A_i, denoted as S_{mid}. Obviously, for tasks in S_h, $[\mathrm{ar}_i, \mathrm{Response}_i]$ conforms to the above two hypotheses: ① when calculating the response time of A_i, it is assumed that all tasks in S_h release at the maximum frequency; ② at $\mathrm{Response}_i$, all instances generated by these tasks have finished. Therefore, for each task in S_h, among all instances of this task with releasing time greater than $\mathrm{Response}_i$, the one with the minimum releasing time is set to S_h^p. When $\mathrm{ar}_i < \mathrm{ar}_h^k + T_h$, the arrival time of S_h^p is

$$\mathrm{ar}_h^k + T_h + \max\left\{0, \left\lceil \frac{\mathrm{Response}_i - \mathrm{ar}_h^k - T_h}{T_h} \right\rceil \cdot T_h \right\}.$$

When $\mathrm{ar}_i \geqslant \mathrm{ar}_h^k + T_h$ or $\mathrm{ar}_h^k = -1$, this time is $\mathrm{ar}_i + \left\lceil \frac{\mathrm{Response}_i - \mathrm{ar}_i}{T_h} \right\rceil \cdot T_h$, where ar_h^k is the releasing time of the last instance of S_h compared to the arrival time of A_i.

For each task in S_{mid}, the relationship between the arrival time ar_i of A_i and the last instance of this task, denoted as S_{mid}^k, are as follows: ① $\mathrm{ar}_{\mathrm{mid}}^k < \mathrm{ar}_i < \mathrm{ar}_{\mathrm{mid}}^k + T_{\mathrm{mid}}$; ② $\mathrm{ar}_{\mathrm{mid}}^k + T_{\mathrm{mid}} < \mathrm{ar}_i$; ③ $\mathrm{ar}_{\mathrm{mid}}^k = -1$. In any case, only when S_{mid} releases at the maximum frequency can the response time of S_j^{k+1} be maximum. Therefore, the minimum releasing time but greater than $\mathrm{Response}_i$ can also be obtained through above process. However, the difference with S_h is that since the priority of S_{mid} is less than A_i, the remaining execution time E_{mid}^k of S_{mid}^k at ar_i can only be executed later than $\mathrm{Response}_i$, and also preempts S_j^{k+1} within $[\mathrm{Response}_i, d_{\mathrm{mid}}^{k+1}]$. The sum of the remaining execution time at ar_i for all tasks in S_{mid} can be recorded as t_{mid}. Therefore, the schedulability of S_j^{k+1} can be determined by RTA, and the initial iteration value is the execution time C_j of S_j^{k+1}.

Algorithm 7 RTA method for determining the schedulability of next sporadic instance when aperiodic task arrives

Input: $\boldsymbol{T}_F(n) = (F_1, F_2, \ldots, F_n)$, $\boldsymbol{T}_s(n) = (S_1, S_2, \ldots, S_n)$, $\boldsymbol{T}_A(n) = (A_1, A_2, \ldots, A_n)$
Output: schedulability of next sporadic instance

1: **bool** schedulable = true
2: **int** current_iteration = C_j
3: **int** release_time(n)
4: **int** last_iteration = 0
5: **for** S_b in $\{S_h \cup S_{\text{mid}}\}$
6: **if** $\text{ar}_b^k \neq -1$ and $\text{ar}_b < \text{ar}_b^k + T_b$
7: release_time(b) = $\text{ar}_b^k + T_b + \max\left\{0, \left\lceil \frac{\text{Response}_i - \text{ar}_b^k - T_b}{T_b} \right\rceil \cdot T_b\right\}$;
8: **if** $\text{ar}_h^k = -1 \left\| (\text{ar}_b^k \neq -1 \text{ and } \text{ar}_i \geqslant \text{ar}_b^k + T_b)\right.$
9: release_time(b) = $\text{ar}_i + \left\lceil \frac{\text{Response}_i - \text{ar}_i}{T_b} \right\rceil \cdot T_b$;
10: **while** current_iteration != last_iteration
11: last_iteration = current_iteration;
12: current_iteration = $C_j + \sum_{A_q \in \boldsymbol{T}_{\text{mid}}(n)} E_q + t_{\text{mid}} + \text{RT}_F(\text{Response}_i, \text{Response}_i + \text{last\_iteration})$;
13: **for** S_b in $\{S_h \cup S_{\text{mid}}\}$
14: **if** (Response$_i$ + last_iteraton > release_time(b)
15: current_iteration += $\left\lceil \frac{\text{Response}_i + \text{last\_iteration} - \text{release\_time}(b)}{T_b} \right\rceil \cdot C_b$;
16: **if** current_iteration > $d_j^k - \text{Response}_i$
17: schedulable = false;
18: **return** schedulable

Note: Obviously, both IBF and RTA require traversing all types of tasks, so the time complexity depends on the maximum number of tasks with higher priority among three types of tasks. When using RTA, an additional array is required to store the period information of sporadic tasks, and its space consumption depends on the number of such tasks.

6 Simulation Experiment and Result Analysis

6.1 Experimental methods and results

We compare the "schedulable ratio" (one of the real-time performance metrics for scheduling algorithms) between our methods (denoted as track-IBF-S and track-RTA-S) and other existing methods. The schedulable ratios of different methods are compared under varying system loads (the utilization ratio of a task set, which is equal to the sum of the utilization ratios of individual tasks).

(1) Experiment 1. Experiment 1 compares the difference in schedulable ratio between our method and VZOP[25] during the offline phase of system, when only fixed-point tasks and sporadic tasks are considered. The initial number of tasks is set to 15, which includes five fixed-point tasks and ten sporadic tasks. For each task, we use UUnifast algorithm[31] to generate the task's parameters. Because VZOP is only applicable to implicit-deadline sporadic task, the relative deadline is not be considered in Experiment 1. The default deadline for sporadic tasks is equal to their periods. The experimental result is shown in Fig. 13.

(2) Experiment 2. Experiment 2 simulates the runtime of system. We add new tasks to system at random instants, then compare the schedulable ratio of task set using OPM and our method under different system load (here the "system load" refers to the utilization ratio of task set in offline phase, and the addition of new tasks will increase this value). We generate relative deadline for each task based on tasks in Experiment 1, so each task changes from implicit deadline to constrained deadline. For every system load, three sporadic tasks and two aperiodic tasks were generated as new tasks that arrive randomly during runtime. Only when these five tasks all pass the schedulability determination, the entire task set is confirmed to be scheduled. Improvement of schedulable ratio by our method is shown in Fig. 14.

6.2 Analysis of experimental results

It can be seen from Fig. 13 that our methods, whether track-IBF-S or track-RTA-S, achieve a significant

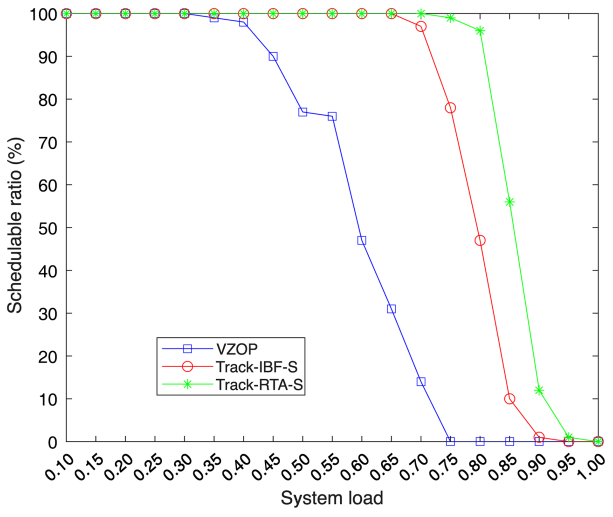


Fig. 13 Comparison of schedulable ratio for different algorithms during offline stage.

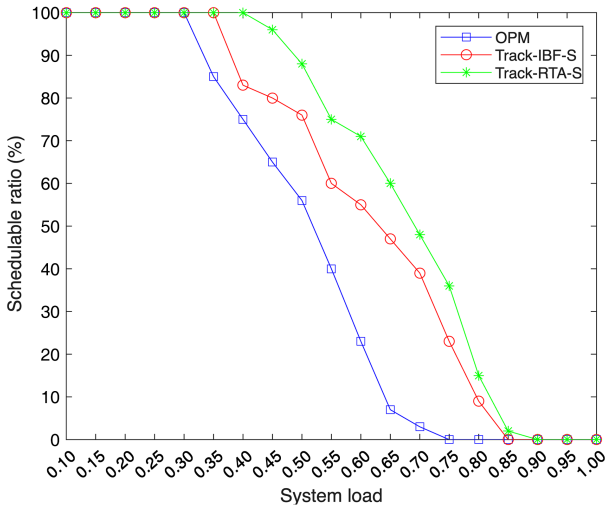


Fig. 14 Comparison of schedulable ratio for different algorithms during online stage.

improvement in schedulable ratio compared to VZOP. This is because VZOP utilizes the upper bound of RM algorithm: $n\left(2^{\frac{1}{n}}-1\right)$, where n is the number of tasks. Therefore, when the number of tasks is fixed, the value of this formula is also determined, meaning that all task sets with utilization ratio greater than this value will be judged as unschedulable. To improve real-time performance, VZOP introduces an additional "false" utilization ratio of the task set, making the overall utilization ratio higher than the actual ratio, which further exacerbates the pessimism. In Experiment 1, the corresponding upper bound of VZOP is 0.715. When the utilization ratio of the task set is greater than or equal to 0.5, the schedulable ratio of VZOP decreases significantly. When the system load is high (greater than 0.7), the schedulable ratio of VZOP is almost 0, while track-IBF-S and track-RTA-S can still maintain a high level. The average schedulable ratio is increased by 38.80% and 51.9%, respectively by track-IBF-S and track-RTA-S when the system load is greater than 0.5.

For Experiment 2, due to the addition of five new tasks compared to offline phase, the overall utilization ratio of task set increases, resulting in a certain decrease in schedulable ratio compared to Experiment 1. As mentioned earlier, OPM has significant pessimism. As system load increases, the schedulable ratio of OPM decreases rapidly. When the utilization ratio of a task set is greater than or equal to 0.35, the average schedulable ratio of track-IBF-S is 15.6% higher than that of OPM, and that of track-RTA-S is 24.1% higher. Further, track-RTA-S always has a higher schedulable ratio than track-IBF-S due to the difference in accuracy between RTA and IBF. When system load is high, track-RTA-S can enable more task sets to be determined as schedulable, which may help better utilize processor resources.

7 Conclusion

We first elaborate on the development background and existing challenges of spacecraft as a typical real-time system. Its characteristics of openness, hard real-time, and supporting multiple types of tasks introduce challenges for task scheduling. Then we select the schedulability determination problem of sporadic tasks with more specific timing constrains in spaceborne operating system. We construct basic but essential system assumptions, task models, and scheduling strategies. The dynamic parameters in task model play an important role in improving the accuracy of the schedulability determination algorithm. On the above basis, we conduct a fine-grained analysis by classifying different preemption situations. RTA and IBF combined with the task's dynamic parameters are respectively used to complete schedulability determination. We achieve a higher analytical accuracy by using RTA, while IBF has a lower time overhead, thus providing flexible choices for system design. Both methods achieve a better schedulable ratio compared with other methods we currently know.

Acknowledgment

This work was supported in part by the State Key Program of National Natural Science Foundation of China (No. 62032004).

References

[1] H. Li, L. Zhang, T. Xiao, and J. Dong, Real-time control for CPS of digital airplane assembly with robust H-infinity theory, *Tsinghua Science and Technology*, vol. 20, no. 4, pp. 376–384, 2015.

[2] T. Cheng, L. Liu, Z. Li, and S. Heng, Real-time dwell scheduling based on a unified pulse interleaving framework for phased array radar, *Tsinghua Science and Technology*, vol. 29, no. 5, pp. 1540–1553, 2024.

[3] A. Giatsintov, K. Mamrosenko, and P. Bazhenov, Architecture of the graphics system for embedded real-time operating systems, *Tsinghua Science and Technology*, vol. 28, no. 3, pp. 541–551, 2023.

[4] P. Rogers, Review of the book: Real-time systems and programming languages (4th edition) by Alan Burns and Andy Wellings, *ACM SIGAda Ada Lett.*, vol. 29, no. 2, p. 71, 2009.

[5] Z. Y. Tang, F. P. Zhe, and X. D. Tang, *Computer*

Operating System, (in Chinese). Xi'an, China: Xidian University Press, 2001.

[6] J. Gong, M. F. Yang, L. Qiao, H. Yang, B. Gu, F. Peng, J. Wang, J. Xu, and B. Liu, Timed scheduling method for tasks in operating system through preemptive priority and round-robin, (in Chinese), China Patent CN103713948A, April 09, 2014.

[7] X. Chen, W. Gu, G. Dai, L. Xing, T. Tian, W. Luo, S. Cheng, and M. Zhou, Data-driven collaborative scheduling method for multi-satellite data-transmission, *Tsinghua Science and Technology*, vol. 29, no. 5, pp. 1463–1480, 2024.

[8] A. K. Rai, N. Mandal, K. K. Singh, and I. Izonin, Satellite image classification using a hybrid manta ray foraging optimization neural network, *Big Data Mining and Analytics*, vol. 6, no. 1, pp. 44–54, 2023.

[9] C. L. Liu and J. W. Layland, Scheduling algorithms for multiprogramming in a hard-real-time environment, *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[10] J. Lehoczky, L. Sha, and Y. Ding, The rate monotonic scheduling algorithm: Exact characterization and average case behavior, in *Proc. 1989 Real-Time Systems Symp*., Santa Monica, CA, USA, 1989, pp. 166–171.

[11] M. Joseph and P. Pandya, Finding response times in a real-time system, *Comput. J.*, vol. 26, no. 5, pp. 390–395, 1986.

[12] N. Fisher, S. Baruah, and T. Baker, The partitioned scheduling of sporadic tasks according to static-priorities, in *Proc. 18th Euromicro Conf. Real-Time Systems*, Dresden, Germany, 2006, pp. 118–127.

[13] S. Baruah, B. Chattopadhyay, H. Li, and I. Shin, Mixed criticality scheduling on multiprocessors, *Real-Time Syst.*, vol. 50, no. 1, pp. 142–177, 2014.

[14] J. Y. T. Leung and J. Whitehead, On the complexity of fixed-priority scheduling of periodic, real-time tasks, *Perform. Eval.*, vol. 2, no. 4, pp. 237–250, 1982.

[15] H. Liu, C. Xi, L. Qiao, J. Zhang, and M. Yang, A schedulability test for sporadic task DM scheduling based on density upper bound, *IEEE Access*, vol. 10, pp. 12475–12486, 2022.

[16] J. J. Chen, W. H. Huang, and C. Liu, K2U: A general framework from k-point effective schedulability analysis to utilization-based tests, in *Proc. 2015 IEEE Real-Time Systems Symp*., San Antonio, TX, USA, 2015, pp. 107–118.

[17] J. H. Sun, J. C. Sun, N. Guan, and Q. X. Deng, Integer programming approach for schedulability of sporadic real-time systems, (in Chinese), *J. Softw.*, vol. 28, no. 2, pp. 411–428, 2017.

[18] G. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. 4th ed. Cham, Switzerland: Springer, 2024.

[19] B. Sprunt, L. Sha, and J. Lehoczky, Aperiodic task scheduling for Hard-Real-Time systems, *Real-Time Syst.*, vol. 1, no. 1, pp. 27–60, 1989.

[20] T. S. Tia, J. W. S. Liu, and M. Shankar, Algorithms and optimality of scheduling soft aperiodic requests in fixed-priority preemptive systems, *Real-Time Syst.*, vol. 10, no. 1, pp. 23–43, 1996.

[21] B. N. Liu, The research of real time hybrid task scheduling algorithm based on multi-core platform, (in Chinese), Master dissertation, Wuhan University of Technology, Wuhan, China, 2013.

[22] G. Tu, Research on scheduling algorithms of soft real-time system tasks, (in Chinese), Ph.D. dissertation, Huazhong University of Science and Technology, Wuhan, China, 2004.

[23] D. Isovic and G. Fohler, Handling mixed sets of tasks in combined offline and online scheduled real-time systems, *Real-Time Syst.*, vol. 43, no. 3, pp. 296–325, 2009.

[24] M. Coutinho, J. Rufino, and C. Almeida, Response time analysis of asynchronous periodic and sporadic tasks sheduled by a fixed priority preemptive algorithm, in *Proc. 2008 Euromicro Conf. Real-Time Systems*, Prague, Czech Republic, 2008, pp. 156–167.

[25] H. B. Liu, L. Qiao, M. F. Yang, X. Chen, Z. Ma, and S. F. Li, Real-time task scheduling and analysis method based on virtual zoom out period, (in Chinese), *J. Softw.*, vol. 33, no. 9, pp. 3512–3528, 2022.

[26] H. Liu, M. Yang, T. Wang, C. Song, S. Zhu, and X. Chen, A heuristic mixed real-time task allocation of virtual utilization in multi-core processor, *J. Inf. Intell.*, vol. 1, no. 2, pp. 156–177, 2023.

[27] L. Z. Jiang, Optimal design and research on multi-core scheduling mechanism of aCoral embedded operating system, (in Chinese), Master dissertation, University of Electronic Science and Technology of China, Chengdu, China, 2021.

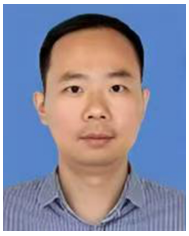
[28] X. Chen, L. Qiao, M. F. Yang, and H. B. Liu, Priority ceiling protocol based on avoidance blocking, (in Chinese), *J. Softw.*, vol. 34, no. 7, pp. 3422–3437, 2023.

[29] H. Jin, H. A. Wang, Q. Wang, and G. Z. Dai, An improved least-slack-first scheduling algorithm, (in Chinese), *J. Softw.*, vol. 15, no. 8, pp. 1116–1123, 2004.

[30] C. H. Song, L. Qiao, M. L. Yang, H. B. Liu, J. J. Jiang, and X. Chen, A two-level admission control strategy for mixed sets of real-time tasks in spaceborne operating system, (in Chinese), *Aerosp. Control Appl.*, vol. 50, no. 2, pp. 93–104, 2024.

[31] E. Bini and G. C. Buttazzo, Measuring the performance of schedulability tests, *Real-Time Syst*., vol. 30, nos. 1&2, pp. 129–154, 2005.

Chenghao Song received the MEng degree from Beijing Institute of Control Engineering, Beijing, China, in 2024. He is currently pursuing the PhD degree at School of Computer Science (National Pilot Software Engineer School), Beijing University of Posts and Telecommunications, Beijing, China. His current research interest includes satellite security and computing.



Lei Qiao received the PhD degree in computer science from University of Science and Technology of China, Hefei, China, in 2007. He is a professor at the Center of On-Board Computer and Electronics, Beijing Institute of Control Engineering, Beijing, China, and Lanzhou University, Lanzhou, China. He is a distinguished member of the CCF. His research interests are in operating system design and formal verification.

Mengfei Yang received the PhD degree from Tsinghua University, Beijing, in 2005. He is currently a researcher and a doctoral supervisor at China Academy of Space Technology, Beijing, China, and an academician of the Chinese Academy of Sciences. His research interest incuding high-speed re-entry of space vehicles, ultra-high-precision attitude control, and highly reliable control computers.

Maolin Yang received the MS degree from Zhejiang University, Hangzhou, China, in 2011, and the PhD degree from University of Electronic Science and Technology of China (UESTC), Chengdu, China, in 2016. He is currently working at School of Information and Software Engineering, UESTC, Chengdu, China. His research interests include scheduling, synchronization, and analysis for embedded real-time operating systems.

Hongbiao Liu received the PhD degree in computer science and technology from Xidian University, Xi'an, China, in 2023. He is an engineer at the Center of On-Board Computer and Electronics, Beijing Institute of Control Engineering, Beijing, China. His research interests are in operating system architecture design and task scheduling technology.

Jun Shen received the PhD degree in computer applications from Southeast University, Nanjing, China, in 2001. He held positions at Swinburne University of Technology, Melbourne, Australia, and University of South Australia, Adelaide, Australia, before 2006. He is currently a full professor at School of Computing and Information Technology, University of Wollongong, Wollongong, Australia. He has published more than 400 papers in journals and conferences in computer science areas. His research interests includes computational intelligence, bioinformatics, and cloud computing.