SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

September 2024, Vol. 67, Iss. 9, 192103:1–192103:17
https://doi.org/10.1007/s11432-022-3891-4

An efficient schedulability analysis based on worst-case interference time for real-time systems

Hongbiao LIU[1], Mengfei YANG[2*], Lei QIAO[3], Xi CHEN[3] & Jian GONG[3]

[1]*School of Computer Science and Technology, Xidian University, Xi'an* 710071*, China;*
[2]*China Academy of Space Technology, Beijing* 100190*, China;*
[3]*Beijing Institute of Control Engineering, Beijing* 100080*, China*

Received 7 April 2022/Revised 2 September 2022/Accepted 10 April 2023/Published online 20 August 2024

Abstract Real-time systems are widely implemented in the Internet of Things (IoT) and safety-critical systems, both of which have generated enormous social value. Aiming at the classic schedulability analysis problem in real-time systems, we proposed an exact Boolean analysis based on interference (EBAI) for schedulability analysis in real-time systems. EBAI is based on worst-case interference time (WCIT), which considers both the release jitter and blocking time of the task. We improved the efficiency of the three existing tests and provided a comprehensive summary of related research results in the field. Abundant experiments were conducted to compare EBAI with other related results. Our evaluation showed that in certain cases, the runtime gain achieved using our analysis method may exceed 73% compared to the state-of-the-art schedulability test. Furthermore, the benefits obtained from our tests grew with the number of tasks, reaching a level suitable for practical application. EBAI is oriented to the five-tuple real-time task model with stronger expression ability and possesses a low runtime overhead. These characteristics make it applicable in various real-time systems such as spacecraft, autonomous vehicles, industrial robots, and traffic command systems.

Keywords five-tuple real-time task model, real-time system, spacecraft, Internet of Things, exact schedulability analysis, worst-case interference time

1 Introduction

With the development of the Internet of Things (IoT), the real-time performance of systems has garnered considerable attention. In the era of IoT, the real-time system plays an increasingly important role [1]. In a real-time system, the correctness of computational tasks depends not only on the logical accuracy of results but also on the time when the calculation results are generated [2]. The real-time operating system (RTOS) serves as the main form of a real-time system, and it has been widely implemented in several domains, including flight control systems, industrial robots, and traffic command systems. These are particularly prevalent in the aerospace field. Therefore, we take RTOSs as the actual subject of consideration. Notably, RTOSs are implemented in the James Webb space telescope, a joint project of the National Aeronautics and Space Administration (NASA), the European Space Agency, and the Canadian Space Agency. Countries and regions represented by the United States, China, Russia, the European Union, Japan, and Canada have used RTOSs in various space exploration missions. High safety-critical real-time tasks (flight control in systems, etc.) in satellite systems such as communications, navigation, remote sensing, and deep-space probes are inseparable from the support of RTOSs. Therefore, the study of the real-time system holds strong academic value and practical significance.

The main challenge in a real-time system is the task of guaranteeing its real-time performance, and a reasonable scheduling strategy is a key technology in achieving this goal. Fixed-priority preemptive scheduling is widely used in real-time systems [3]. Compared with other scheduling algorithms, such as dynamic priority scheduling [4], round-robin scheduling [5], and static table-driven scheduling [6], this scheduling policy offers several advantages, including simplicity of implementation, low runtime

* Corresponding author (email: yangmf@bice.org.cn)

overhead, high real-time performance, strong scalability, and strong predictability. These advantages have established fixed-priority preemptive scheduling as the fundamental scheduling policy in several commercial embedded RTOSs.

Motivation. The spacecraft computer system used in the satellite is a typical robust real-time system operating under a fixed-priority preemptive scheduling policy. Spacecraft feature several hard real-time tasks, such as sensor data acquisition and orbit control. It is imperative that these tasks are completed before their respective deadlines to prevent severe consequences. Therefore, conducting schedulability analysis on these tasks before execution becomes essential in determining whether the deadline is met and ensuring system safety. Compared with general embedded systems, spacecraft have limited resources and operate in more demanding environments [7, 8]. Consequently, scheduling algorithms demand higher efficiency and reliability standards. Successfully addressing the task schedulability analysis problem in aerospace systems can pave the way for its application in other real-time systems. However, schedulability analysis is challenging due to the following two characteristics of the spacecraft computer system.

The number of tasks is subject to dynamic variations. With the advancement of aerospace application requirements (such as Mars exploration and manned missions to the Moon), the system will enable the dynamic addition of tasks through the dynamic task loading mechanism [9], implementing dynamic expansion and updating system functions at runtime [10]. To ensure that all real-time tasks in the system continue to meet their deadlines, it becomes necessary to perform an online schedulability analysis.

Moreover, Tasks may experience jitter release and blocking time for resource access [11]. With the increase in the complexity of system functions, multiple tasks will access shared resources. A widely known example of this is the priority inversion event of NASA's Mars Pathfinder in 1997. In this example, the low-priority meteorological task preemptively occupied the shared resources while the medium-priority communication task continuously preempted the processor. This led to the inability in scheduling the high-priority bus task, leading to an indefinite system reboot. Therefore, it is of great significance to consider the impact of shared resource blocking time and jitter release on task schedulability.

The above features make on-line schedulability analysis challenging in a spacecraft computer system. In a resource-constrained real-time system that uses a fixed-priority preemptive scheduling policy, there is a need for an efficient, accurate, and universal schedulability analysis technique to ensure the system's real-time performance and reliability.

In response to these problems, this study combines the latest research progress comprehensively analyze the key issues. The main research contributions and innovations are summarized as follows:

(1) The exact Boolean analysis based on interference (EBAI), an exact Boolean schedulability test, is proposed. We consider a five-tuple task model that includes release jitter and blocking time for each task. We analyzed the worst-case interference time (WCIT) of higher-priority tasks and combined it with response-time analysis (RTA) to form the EBAI test. The experimental results demonstrate that it is more efficient than other alternatives. In certain cases, our test achieved a runtime gain exceeding 73% compared to other available schedulability tests. Due to the time complexity of WCIT being only $O(n^2)$ and its high precision, WCIT contributes to improving the efficiency of schedulability analysis. Combined with RTA, EBAI can provide an exact schedulability analysis result while keeping the run time overhead low.

(2) The schedulability test efficiency of three existing tests has been improved. By exploring the principles of these schedulability tests, we modified the testing process to remarkably enhance the efficiency of the three schedulability tests. Additionally, the related research results in this field over the years are summarized, and abundant comparisons are made. This is instructive for the practical application of schedulability analysis.

The remainder of the paper is organized as follows. Section 2 provides an introduction to the background and related work. Section 3 describes the system model. Section 4 describes the schedulability test in detail. Section 5 presents the experiment and analysis of the results. Section 6 serves as the conclusion.

2 Background and related work

The research of real-time task systems is mainly divided into scheduling strategies and schedulability analysis. According to the representation ability of the task model, real-time tasks can be mainly divided

into periodic tasks, sporadic tasks and aperiodic tasks. As the most basic task scheduling strategy, fixed-priority preemptive scheduling has been widely studied by the real-time scheduling theory community.

2.1 Task model

For periodic real-time tasks, from [12], it can be seen that the schedulability analysis is a weak NP-complete problem when the total utilization of the task set is an arbitrary value. Liu et al. [13] proposed the rate-monotonic (RM) scheduling. The priority assigned to a task depends upon its period. The smaller the period of tasks is, the greater the priority of tasks will be assigned (with ties broken arbitrarily), and vice versa. The RM scheduling is proven to be the optimal scheduling algorithm in the fixed-priority preemptive scheduling systems upon uniprocessors by switching tasks in pairs. A tight utilization bound for a task set scheduled by RM is given, and it is a sufficient schedulability test with time complexity $O(n)$, where n, C_i, D_i are the number of tasks, worst-case execution time (WCET) and period of task i, respectively.

$$\sum_{i=1}^{n}(C_i/T_i) \leqslant n(\sqrt[n]{2}-1). \tag{1}$$

Later, Bini et al. [14] proposed using the hyperbolic bound to determine the schedulability of the tasks scheduled by RM and proved that this bound is tighter than Liu, where U_i is the utilization of task i.

$$\prod_{i=1}^{n}(1+U_i) \leqslant 2. \tag{2}$$

In the same year, Lauzac et al. [15] proposed a RBound test that considered both the number of tasks and the ratio of the maximum period to the minimum period in the task set, and also proved that this bound is tighter than Liu, where k is this ratio.

$$\sum_{i=1}^{n}(C_i/T_i) \leqslant (n-1)(k^{1/(n-1)}-1)+2/k-1. \tag{3}$$

However, the schedulability tests above are only sufficient, which means tasks may be still schedulable even if they do not meet the bound condition. Combining the critical time zones, Lehoczky et al. [16] proposed an efficient schedulability test and it can exactly determine the schedulability of the periodic tasks scheduled by the RM. In this test, one only needs to check whether there is a schedulable time point that can meet the processor demand in an integer multiple time of the period ratio. This test greatly reduces the number of time points to be checked, reducing the time complexity of RM scheduling to $O(n^3)\times(T_{\max}/T_{\min})$, where $T_{\max}$ is the maximum period of the task set, and $T_{\min}$ is the minimum period. However, this schedulability test does not consider the impact of the order of checkpoints on the test efficiency. Therefore, we make an improvement in our paper that further speeds up this schedulability test. Since all test points are less than D_i and combine with Theorem 1 in [17], the test can start directly from the largest point. If the processor demand is less than D_i, the task can be determined to be schedulable, otherwise continue testing at smaller integer multiples points. We can speed up the efficiency of the schedulability test through this strategy. The improved points are (4), and the comparison with the experimental results before the improvement is performed in Experiment 8. It can be seen from Experiment 8 that the efficiency of the schedulability test has been greatly improved.

$$S_i=\{kT_j|j=1,2,\ldots,i;k=\lfloor T_i/T_j\rfloor,\ldots,2,1\}. \tag{4}$$

Regarding the priority assignment of the asynchronous period tasks, Audsley [18] proposed the optimal fixed-priority assignment algorithm optimal priority assignment (OPA). The OPA of a task set can be obtained in no more than $n(n+1)/2$ times schedulability tests. However, periodic tasks scheduled by RM contain many basic assumptions: all tasks are periodic, released at the beginning of the period, and have a deadline equal to the period, etc. However, the tasks in practical applications may not satisfy these basic assumptions, making the practical application of periodic tasks limited. Therefore, more research is aimed at the scheduling problem of sporadic and aperiodic tasks.

For sporadic real-time tasks, Leung et al. [19] first proposed the deadline monotonic (DM) scheduling algorithm. The priority assigned to a task depends upon its deadline. Tasks with smaller deadlines

are assigned higher priorities (with ties broken arbitrarily), and vice versa. DM scheduling has been proven to be the optimal scheduling algorithm for scheduling sporadic tasks that share a critical instant. Liu et al. [20] analyzed the density upper bound of the sporadic task set under DM scheduling and provided a sufficient schedulability test. For the aperiodic task, Abdelzaher et al. [21] proposed that a sufficient determination formula under DM scheduling is (5). Since the sporadic task can be seen as a special pattern of the aperiodic task, this formula can also be used as a schedulability test for sporadic tasks. They proved that it represents the optimal upper bound for the set of aperiodic tasks under DM scheduling, where D_i is the deadline of task i.

$$\sum_{i=1}^{n}(C_i/D_i) \leqslant 1/(1+\sqrt{1/2}). \tag{5}$$

However, in practical applications, sporadic tasks may be assigned to arbitrary priorities instead of following DM scheduling, which further complicates the schedulability analysis.

2.2 Schedulability test

The schedulability analysis of sporadic real-time tasks for fixed-priority preemptive scheduling on uniprocessors is an intractable problem, which has been proven by Ekberg et al. [22] to be Weakly NP-complete. Bonifaci et al. [23] discussed the special case of task systems with the harmonic period, and gave an exact schedulability test with polynomial time complexity. Chen et al. [24] proposed to determine the schedulability of the tasks by verifying the hyperbolic formula, as (6), where $hp_1(k)$ indicates that a set of tasks with priority is higher than τ_k and period is less than D_k. $hp_2(k)$ represents a set of tasks with priority is higher than τ_k and period is greater than or equal to D_k.

$$\left[\left(C_k+\sum_{\tau_i\in hp_2(k)}C_i\right)\Big/D_k+1\right]\prod\nolimits_{\tau_i\in hp_1(k)}(1+U_i)\leqslant 2. \tag{6}$$

The time complexity is only $O(n^2)$, but this is only a sufficient condition for the schedulability test. Due to the problems of shared resources and kernel overhead in actual RTOSs, sporadic tasks may have blocking time and release jitter. In such a situation, the above analysis methods cannot be performed. We need to find a more general approach for schedulability analysis. The schedulability test described next is able to analyze the schedulability of sporadic real-time tasks with blocking time [25] and release jitter [26].

RTA [27, 28] is a more general schedulability test proposed by Joseph, Pandya and Audsley, which can exactly determine the schedulability of the sporadic tasks rather than only a sufficient condition. It calculates the worst-case response time (WCRT) R_i of the task i, and compares it with the task's deadline to determine the schedulability of the task. The calculation formula of WCRT is

$$R_i = C_i + \sum_{j=1}^{i-1}\lceil R_i/T_i\rceil C_j,$$

where C_i is the execution time and T_i is the period of task i, and the priority of task j is lower than task i. This is a recursive formula, and moreover, it contains a ceil operation. Hence the analytical solution of WCRT is difficult to solve. However, we can solve it numerically in an iterative manner. We first need to select an initial value and then start iterating. The iteration ends when R_i converges to a fixed point or it is greater than the deadline of task i. This fixed point is the result of WCRT for the task i. However, the problem can still be intractable if the parameters involved in the RTA process are rational, it is an NP-Hard problem [29]. Even if the parameters are natural numbers, the time complexity is $O(n^2\times D_{\max})$, where $D_{\max}$ is the maximum deadline of the task set [30]. The difficulty in RTA is how to quickly calculate WCRT, so the follow-up research results are to improve the calculation process.

2.3 Initial value of schedulability test

In order to speed up the calculation of WCRT, researchers mainly find a larger initial value to reduce the number of iterations. As can be seen from Figure 1 in [17], there are two fixed points in the iteration

of RTA, and the first fixed point is the correct value of WCRT. If the selected initial value of iteration is smaller than the first fixed point, according to Theorem 1 of [17], it must be able to converge to the correct WCRT value, called an exact initial value. However, if the selected initial value of iteration is greater than the first fixed point, the convergence result is greater than the WCRT, which is called a sufficient initial value and it is still effective for a sufficient schedulability test.

Regarding the exact initial value of RTA, the first one is the WCET of the task, and it opened a new wave of initial value research, as (7), where B_i is the blocking time of task i.

$$R_i^0 = C_i + B_i. \tag{7}$$

Later, Hansson et al. [17] and Bril et al. [31] calculated the lower bound of the response time and proposed an exact initial value, as (8), where J_j is the release jitter of task j.

$$R_i^0 = R_i^{\mathrm{LB}} = \left(C_i + B_i + \sum_{j<i} J_j U_j \right) \Big/ \left(1 - \sum_{j<i} U_j \right). \tag{8}$$

Since this initial value needs to calculate the cumulative utilization of higher-priority tasks and the number of operations changes with i, a lot of calculations will be needed. Hence we make an improvement for its calculation process in our paper to reduce the calculation overhead. We improve the efficiency by only calculating $J_{i-1}U_{i-1}$ and U_{i-1} each time, and add them to the history accumulation sum. The experiments of test efficiency which compare the method of combining the history accumulation sum and the uncombined one are shown in Experiment 9. It is obvious that the efficiency of combining the historical sum method is higher. In addition, they analyzed the relationship among tasks' response time and proposed another more effective exact initial value in (9), where R_{i-1} is the response time of the task τ_{i-1}. Therefore, the schedulability test is required to proceed sequentially from the high priority task to the low priority task.

$$R_i^0 = R_{i-1} + C_i. \tag{9}$$

Later, Lu et al. [32] considered the relationship between task periods and proposed a composite exact initial value $\max\{T_i - T_{i-1}, T_i/2\}$, where T_i is the period of task i. Their result shows that this initial value may achieve a saving of up to 78.2% in the number of iterations. When the number of tasks in a system is large, it can significantly reduce the number of iterations. Inspired by this initial value, Davis et al. [33] extended it to include release jitter J_i and blocking time B_i, and obtained an improved initial value in

$$R_i^0 = (D_i - J_i + C_i + B_i)/2. \tag{10}$$

They proved that it is optimal in the sense that the initial value is tight for RTA and any increase of this initial value will make this test convert to a sufficient test. Since then, no one studied the exact initial value anymore. Later, the main research is about the efficiency of sufficient initial value.

The sufficient initial value is first used to test the schedulability of the task set. If it fails to pass the test, then the exact initial value is used. This strategy can further improve the efficiency of the schedulability test. Based on [34], Davis et al. [35] analyzed the upper bound of response time and proposed a sufficient initial value, which produces an exact Boolean schedulability test. The upper bound is

$$R_i^0 = R_i^{\mathrm{UB}} = \left\{ C_i + B_i + \sum_{j<i} [J_j U_j + C_j(1 - U_j)] \right\} \Big/ \left(1 - \sum_{j<i} U_j \right). \tag{11}$$

Similar to the initial value of (8), we still use the strategy of combining historical accumulation to improve the efficiency of this schedulability test. The experiments of test efficiency comparison are shown in Experiment 9. It is obvious that the efficiency of combining the historical sum method is higher. Yu et al. [36] proposed an adjustment parameter δ to minimize the number of iterations of the RTA test.

$$R_i^0 = (D_i - J_i + C_i + B_i)\delta. \tag{12}$$

The experiments showed that $\delta = 0.9$ minimized the number of iterations, and improved the efficiency of the schedulability test. Other related researches about the schedulability test of the sporadic task are mainly about self-suspended task model [37], direct acyclic graph (DAG) task model [38], hybrid task scheduling [39], and multi-core scheduling [40].

Table 1 Comparison of schedulability tests

Short name	Identfier	Precision	Boolean	Time complexity	Task model	Priority assignment	Blocking and release time
EBAI (this paper)	Alg. 1	Exact	✓	NP-Hard	Sporadic	Fixed-priority	✓
Liu	Eq. (1)	Sufficient	✓	$O(n)$	Periodic	RM	×
Bini	Eq. (2)	Sufficient	✓	$O(n)$	Periodic	RM	×
Lauzac	Eq. (3)	Sufficient	✓	$O(n)$	Periodic	RM	×
Imp Lehoczky (this paper)	Eq. (4)	Exact	✓	$O(n^3) \times (T_{\max}/T_{\min})$	Periodic	RM	×
Hyperbolic	Eq. (6)	Sufficient	✓	$O(n^2)$	Sporadic	Fixed-priority	×
Abdelzaher	Eq. (5)	Sufficient	✓	$O(n)$	Aperiodic	DM	×
RTA	Eq. (7)	Exact	×	NP-Hard	Sporadic	Fixed-priority	✓
Imp Hansson (this paper)	Eq. (8)	exact	×	NP-Hard	Sporadic	Fixed-priority	✓
RTA Seq	Eq. (9)	Exact	×	NP-Hard	Sporadic	Fixed-priority	✓
RTA optimal	Eq. (10)	Exact	×	NP-Hard	Sporadic	Fixed-priority	✓
Imp Davis (this paper)	Eq. (11)	Exact	✓	NP-Hard	Sporadic	Fixed-priority	✓
Yu	Eq. (12)	Exact	✓	NP-Hard	Sporadic	Fixed-priority	✓

2.4 State of research

Existing schedulability tests can be divided into three groups: periodic task research group, sporadic task research group, and aperiodic task research group. These schedulability tests can be summarized in Table 1.

Although the schedulability analysis theory has been developed for many years and has made great progress, the above methods still have two problems. On the one hand, the sufficient schedulability tests have low time complexity, but they cannot exactly give the schedulability of the tasks, which makes the processor underutilized; on the other hand, the exact schedulability tests still have a high run time overhead, which results in a limited range of practical applications.

3 System model

We consider the schedulability test of sporadic tasks which include release jitter and blocking time under uniprocessor. It is a five-tuple task model, $\tau_i = (C_i, D_i, T_i, J_i, B_i)$, where satisfies $0 < C_i \leqslant D_i \leqslant T_i, J_i \leqslant T_i, B_i \leqslant T_i$ and $C_i, D_i, T_i, J_i, B_i \in R$, as shown in Figure 1.

C_i represents the WCET of the task τ_i (C_i consists of two parts, $C_i = C_i^1 + C_i^2$). D_i represents the relative deadline of τ_i. One arrival of a task is called an instance (also called job) and the instance arrives at any time, but there is a minimum arrival time interval T_i between the two instances. In the worst-case, it can be regarded as the period of τ_i. τ_i^j represents the jth instance of τ_i. τ_i^{j+1} represents the $(j+1)$th instance of τ_i. The difference between the arrival time of τ_i^j and the arrival time of τ_i^{j+1} is greater than or equal to T_i, where $j \in N$ and N is the set of all integers. J_i represents the release jitter time of τ_i, which represents the maximum time difference between the release time and the arrival time of τ_i. B_i represents the worst-case blocking time of τ_i. In the case of shared resource access, higher-priority tasks may be blocked by lower-priority tasks. Under the real-time locking protocols [41], B_i is the worst-case blocking time. The set of real-time tasks is denoted as $S_H = \{\tau_1, \tau_2, \ldots, \tau_n\}$, where n represents the task number of S_H.

The task index is arranged in order of priority from high to low, the priority of τ_j is higher than τ_i if and only if $j < i$.

The response time represents the time difference between the task's finish and release. Let R_i denote the response time of τ_i. Let Δ denote the sum of the processor demand of all higher-priority tasks $\{\tau_j\}$ in the interval $[0, R_i]$, $j < i$, so

$$R_i = C_i + B_i + \Delta. \tag{13}$$

τ_i is called schedulable, which means that each instance of τ_i can be completed before the deadline, that is

$$R_i \leqslant D_i - J_i \leftrightarrow \tau_i \text{ is schedulable.} \tag{14}$$

The real-time task set S_H is called schedulable, which means that all tasks in S_H are schedulable, that is, $\forall \tau_i \in S_H \rightarrow \tau_i$ is schedulable.

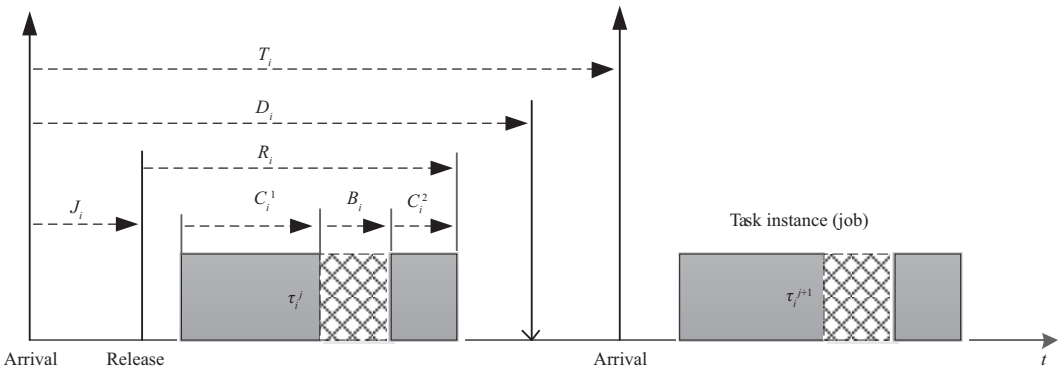

Figure 1 Five-tuple task model.

The utilization of τ_i is expressed as $u_i = C_i/T_i$, and the utilization of S_H is expressed as $U = \sum u_i, \tau_i \in S_H$.

The schedulability analysis refers to obtaining information of task set such as schedulability through the mathematical analysis. The schedulability test is the process of using the schedulability analysis method to determine whether the task set is schedulable and it is also called schedulability determination.

If a schedulability test can only give the schedulability result but not the WCRT, such a test is called a Boolean test.

The performance metric of a schedulability test is the schedulable ratio and run time overhead. A more efficient schedulability test means that compared with other tests, its run time overhead is lower when the schedulable ratio is the same. Alternatively, when the run time overhead is the same, the schedulable ratio is higher.

4 Schedulability analysis

We will derive an efficient exact schedulability test in this section. Firstly, we analyze the possible maximum interference time of the task may suffer, and obtain an efficient sufficient condition for schedulability test. Then we combine this condition with the exact test RTA to form the final schedulability test. Finally, we prove its correctness through the strategy of contradiction.

4.1 Basic concepts

Definition 1 (Analysis interval I_i)**.** It represents the interval from the release time of a τ_i' job to its deadline.

Definition 2 (Worst-case interference time $\mathrm{WCIT}_{j,i}(L)$)**.** It means that the maximum interference time of higher-priority task τ_j on τ_i in an interval (contains τ_j's release times) of length L.

As shown in Figure 2, when $L = D_i$, due to the existence of the release jitter J_j of τ_j, the release interval of the τ_j's job is less than the period T_j. Therefore, in the analysis interval I_i, the higher-priority task τ_j causes greater interference to τ_i. The length of the interval I_i is D_i, so τ_j will release $(D_i + J_j)/T_j$ times complete jobs, resulting in interference time $\lfloor (D_i + J_j)/T_j \rfloor C_j$. Moreover, an incomplete job(carry-out job) is contained in the remaining interval whose length is $(D_i + J_j) \bmod T_j$, which may be greater than C_j. Actually, the interference time caused by this incomplete job should be the minimum of the two. That is $\min\{C_j, (D_i + J_j) \bmod T_j\}$, where $j < i$, mod represents the remainder. Therefore, in the analysis interval I_i, the maximum interference time that the higher-priority task τ_j may cause to τ_i is

$$\mathrm{WCIT}_{j,i}(D_i) = \lfloor (D_i + J_j)/T_j \rfloor C_j + \min\{C_j, (D_i + J_j) \bmod T_j\}. \tag{15}$$

Definition 3 (Actual interference time $\mathrm{AIT}_{j,i}(L)$)**.** It means that the actual interference time of higher-priority task τ_j on τ_i in an interval (contains τ_j's release times) of length L.

In the actual scheduling pattern, the interference time of higher-priority task τ_j is less than or equal to the WCIT $\mathrm{WCIT}_{j,i}(D_i)$. This is because the carry-out jobs of these higher-priority tasks $\{\tau_j, j < i\}$ may overlap in the interval I_i, and a uniprocessor can only execute one carry-out job at a time, resulting

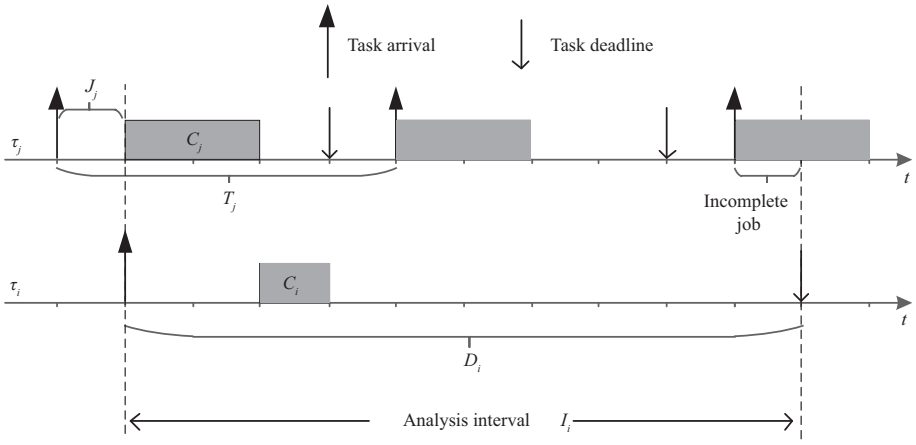

Figure 2 WCIT of τ_j on a job of τ_i.

in the actual interference time being less than or equal to $\mathrm{WCIT}_{j,i}(D_i)$ in I_i, as expressed in (16). The following example will show this situation:

$$\mathrm{AIT}_{j,i}(L) \leqslant \mathrm{WCIT}_{j,i}(L). \tag{16}$$

Example 1. Task set S_H contains 3 tasks, where $\tau_1 = (2, 4, 8, 1, 0), \tau_2 = (1, 4, 7, 0, 0), \tau_3 = (4, 8, 9, 0, 0)$, as shown in Figure 3. The release time of tasks is independent of each other and τ_1, τ_2, τ_3 are released simultaneously. Therefore, it is the worst-case preemption situation for τ_3 [13]. For the schedulability analysis of τ_3, $I_3 = [1, 9]$ is taken as the analysis interval and the length of I_3 is 8. We can calculate the maximum interference of τ_1 on τ_3 is $\mathrm{WCIT}_{1,3}(8) = \lfloor (8+1)/8 \rfloor \times 2 + \min\{2, (8+1) \bmod 8\} = 3$ and τ_2 on τ_3 is $\mathrm{WCIT}_{2,3}(8) = \lfloor (8+0)/7 \rfloor \times 1 + \min\{1, (8+0) \bmod 7\} = 2$. We can see that the processor is executing a higher-priority task τ_1 in the interval $[8, 9]$, hence the incomplete job of τ_2 cannot produce interference time on τ_3. The actual interference time of τ_2 on τ_3 is only $\mathrm{AIT}_{2,3}(D_3) = 1$ which is less than $\mathrm{WCIT}_{2,3}(D_3)$. This is because in the interval $[8, 9]$, the processor is executing higher-priority task τ_1. So the incomplete job of τ_2 cannot interfere with τ_3 and the actual interference time of τ_2 to τ_3 is less than the maximum interference time $\mathrm{WCIT}_{2,3}(8)$.

Combined with the definition of analysis interval and WCIT above, we can obtain Lemma 1.

Lemma 1. If task τ_i is not schedulable, $C_i + B_i + J_i + \sum_{j<i} \mathrm{WCIT}_{j,i}(D_i) > D_i$ will hold.

Proof. Since τ_i is not schedulable, there must be a job of τ_i misses its deadline. The reason this job misses its deadline is that the sum of the actual interference time of higher priority tasks $\{\tau_j, j < i\}$ on this job plus C_i, J_i and blocking time B_i from lower-priority tasks $\{\tau_k, k > i\}$ is greater than this job's deadline D_i. This can be expressed as

$$C_i + B_i + J_i + \sum_{j<i} \mathrm{AIT}_{j,i}(D_i) > D_i. \tag{17}$$

Combining (16) and (17), we can obtained the following equation:

$$C_i + B_i + J_i + \sum_{j<i} \mathrm{WCIT}_{j,i}(D_i) > D_i. \tag{18}$$

Therefore, if the task τ_i is not schedulable, Eq. (18) will hold. The proof is completed.

According to the relationship between the original proposition and the inverse negation proposition, we can obtain a sufficient condition for the task τ_i to be schedulable.

Theorem 1. If $C_i + B_i + J_i + \sum_{j<i} \mathrm{WCIT}_{j,i}(D_i) \leqslant D_i$, then the task τ_i must be schedulable.

Proof. According to Lemma 1, its inverse negation proposition is $C_i + B_i + J_i + \sum_{j<i} \mathrm{WCIT}_{j,i}(D_i) \leqslant D_i$, then the task τ_i must be schedulable. The proof is completed.

Example 2. Task set S_H contains 4 tasks, where $\tau_1 = (2, 4, 8, 1, 0), \tau_2 = (1, 4, 7, 0, 0), \tau_3 = (3, 8, 9, 0, 1)$, $\tau_4 = (1, 10, 11, 0, 0)$. The release time of tasks are independent of each other. τ_1, τ_2, τ_3 are released simultaneously and τ_4 will be released at any small time ε in advance, $\varepsilon \to 0$. τ_4 locks the shared resources in advance, so τ_3 will experience the largest interference and blocking time, as shown in Figure 4. τ_3

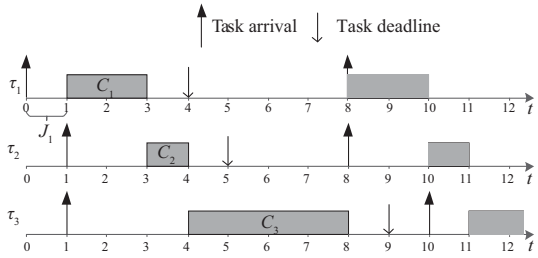

Figure 3 Example of WCIT.

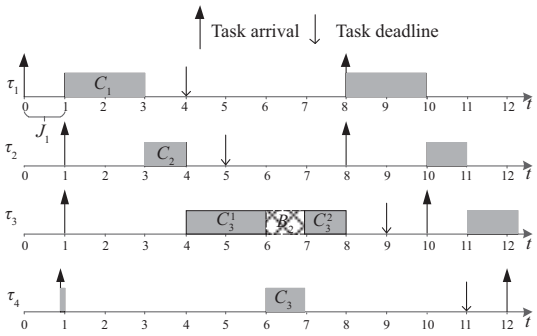

Figure 4 Example of sufficient conditions for schedulability test.

Algorithm 1 Bool EBAI(S_H)

Require: Real-time task set S_H.
Ensure: The schedulability of S_H.

for $\tau_i : S_H$ **do**
  for $(j = 0; j < i; ++j)$ **do**
    $\mathrm{WCIT}_{j,i}(D_i) = \mathrm{floor}((D_i + J_j)/T_j) \times C_j + \min(C_j, (D_i + J_j) \bmod T_j)$;
  end for
  if $C_i + B_i + J_i + \sum_{j<i} \mathrm{WCIT}_{j,i}(D_i) > D_i$ **then**
    $R_i^1 = (D_i - J_i + C_i + B_i)/2$;
    $R_i^2 = 0$;
    while (1) **do**
      Sum = 0;
      for $j = 0; j < i; ++j$ **do**
        Sum+ = ceiling$((R_1 + J_i)/T_i) \times C_i$;
      end for
      $R_2 = C_i + B_i +$ Sum;
      if $R_1 < R_2$ **then**
        if $R_2 > (D_i - J_i)$ **then**
          Return false;
        end if
        $R_1 = R_2$;
      else
        Break;
      end if
    end while
  end if
end for
Return true.

arrives at time 1 and finishes at time 8. However, we can calculate $C_i + B_i + J_i + \sum_{j<i} \mathrm{WCIT}_{j,i}(D_i) = 3 + 1 + 0 + \sum_{j<3} \mathrm{WCIT}_{j,i}(8) = 9$, which is greater than D_3. Therefore, according to Theorem 1, we cannot determine the schedulability of S_H. Theorem 1 is only a sufficient condition for determining the schedulability of a task set. However, Theorem 1 has an extremely low time complexity, and the schedulable ratio is very high, which we will see in Experiment 10.

We combine Theorem 1 with the RTA test to obtain a more efficient exact Boolean schedulability test EBAI.

4.2 Schedulability test step

Real-time task set $S_H = \{\tau_1, \tau_2, \ldots, \tau_n\}$, $\tau_i = (C_i, D_i, T_i, J_i, B_i)$, $\forall \tau_i \in S_H$, $\pi_1 > \pi_2 > \cdots > \pi_n$, where π_i represents the priority of τ_i, the larger π_i, the higher the priority of task τ_i. The steps of the EBAI test which can exactly determine the schedulability of S_H are as follows, as shown in Algorithm 1.

Step 1. Select τ_i in S_H from first to last. If the selection ends, go to step 5, otherwise go to step 2.

Step 2. Calculate the maximum sum of interference to τ_i caused by all higher-priority tasks is $\sum_{j<i} \mathrm{WCIT}_{j,i}(D_i)$. If $C_i + B_i + J_i + \sum_{j<i} \mathrm{WCIT}_{j,i}(D_i)$ is greater than D_i, go to step 3, otherwise go to step 1.

Step 3. Perform exact test RTA on τ_i to get the WCRT R_i, where the initial value is equal to the optimal initial value [33] of the RTA test, $(D_i - J_i + C_i + B_i)/2$.

Step 4. If R_i is greater than D_i, return false and the test ends.

Step 5. Indicate that $\forall \tau_i \in S_H$, τ_i is schedulable, then S_H is schedulable (see Theorem 1), return true, and the test ends.

4.3 Proof of schedulability

Theorem 2. Real-time task set $S_H = \{\tau_1, \tau_2, \ldots, \tau_n\}$, where $\tau_i = (C_i, D_i, T_i, J_i, B_i)$, satisfies $\pi_1 > \pi_2 > \pi_3 > \cdots > \pi_n, 0 < C_i \leqslant D_i \leqslant T_i, J_i \leqslant T_i, B_i \leqslant T_i, C_i, D_i, T_i, J_i, B_i \in R$. EBAI($S_H$) = true, if and only if S_H can be scheduled.

Table 2 Experimental platform

Programming environment	Runtime environment
Programming platform: PC	Operating platform: space station computer
CPU: Intel i5-3470	CPU: BM3803 single core
Main frequency: 3.2 GHz	Main frequency: 80 MHz
RAM: 8 GB	RAM: SDRAM 4 MB
Operating system: Win7	RTOS: SpaceOS 2.0

Proof. Sufficiency. Known from the EBAI test steps, EBAI(S_H) = true, there is $\forall\tau_i \in S_H$, $C_i+B_i+J_i+\sum_{j<i}\mathrm{WCIT}_{j,i}(D_i) \leqslant D_i$ or $R_i \leqslant D_i - J_i$. According to Theorem 1 and (14), $\forall\tau_i \in S_H, \tau_i$ is schedulable, so S_H is schedulable.

Necessity. Prove its inverse negation proposition. If EBAI(S_H) = false, then S_H is unschedulable. Known from the EBAI test steps, there is $\exists\tau_i \in S_H, R_i > D_i - J_i$. According to (14), τ_i is not schedulable, so S_H is not schedulable. The proof is completed.

4.4 Analysis of performance

There are two reasons why WCIT helps improve the efficiency of the schedulability test.

• The time complexity of WCIT is low. The time complexity of WCIT is only $O(n^2)$, where n represents the number of tasks in the task set S_H. The calculation formula of WCIT is simple, and there are no complex operations, hence the run time overhead is very low.

• The schedulable ratio of WCIT is high. WCIT directly calculates the interference of higher-priority tasks, which is more in line with the actual running status of tasks. In (15), the first item is always exact and the pessimism may come from the second item. This is because when multiple carry-out jobs are requested at the same time, the amount of interference will be summed redundantly. However, this makes the schedulability test pessimistic only when the utilization of the task set is high (we can see this in Experiment 10). Therefore, in general, WCIT can make an accurate test, making the schedulable ratio very high.

EBAI is an exact schedulability test combining WCIT and RTA. The time complexity of WCIT is $O(n^2)$ and that of RTA is NP-Hard, hence the time complexity of EBAI is NP-Hard. The space complexity is $O(n)$.

If we do not combine RTA and only use WCIT (Theorem 1) to do the sufficient schedulability test, the comparison of WCIT and EBAI can be seen in Experiments 10–13. We can see that WCIT can be very useful for sufficient testing. After combining RTA, we can obtain the necessary and sufficient schedulability test EBAI and the time overhead does not increase significantly. The next experiments further compare EBAI with other available schedulability tests.

5 Experiment

5.1 Experimental platform

To align with practical applications, we choose the spacecraft computer as our experimental platform. The experimental platform parameters are shown in Table 2.

5.2 Experimental parameters

The performance metrics considered in the experiment are the schedulable ratio r and run time overhead t. To evaluate these metrics, several sporadic real-time task sets are randomly generated during the experiment. The parameters of each experiment are shown in Table 3. Experiments 1–3 compare the schedulable ratio, and Experiments 4–6 compare the run time overhead. Experiments 7–9 involve the improved existing tests mentioned in Section 2, and Experiments 10–13 examine the performance of WCIT mentioned in Subsection 4.4. Some parameters are explained below:

• The number of tasks n. $n = 30$, because the trend of the curve can already be shown.

• Task set utilization U. According to the UUnifast algorithm [42], the utilization of each task is uniformly distributed.

Table 3 Parameters of 13 comparative experiments

Index	n	U	d	J_i	B_i	Priority assignment	Metric	Illustration
Experiment 1	30	$[0.1, 1]$	0.5	✓	✓	Random priority	r	Figure 5(a)
Experiment 2	30	$[0.1, 1]$	0.5	✓	✓	DM priority	r	Figure 5(b)
Experiment 3	30	$[0.1, 1]$	0	×	×	RM priority	r	Figure 5(c)
Experiment 4	$[1, 30]$	0.5	0.5	✓	✓	DM priority	t	Figure 6(a)
Experiment 5	$[1, 30]$	0.5	0	×	×	RM priority	t	Figure 6(b)
Experiment 6	30	$[0.1, 1]$	0	×	×	RM priority	t	Figure 7
Experiment 7	$[1, 30]$	0.5	0	×	×	RM priority	t	Figure 8
Experiment 8	$[1, 30]$	0.5	0	×	×	RM priority	t	Figure 9
Experiment 9	$[1, 30]$	0.5	0	×	×	RM priority	t	Figure 10
Experiment 10	30	$[0.1, 1]$	0.5	✓	✓	DM priority	r	Figure 11(a)
Experiment 11	$[1, 20]$	0.6	0.5	✓	✓	DM priority	t	Figure 11(b)
Experiment 12	$[1, 20]$	0.7	0.5	✓	✓	DM priority	t	Figure 11(c)
Experiment 13	$[1, 20]$	0.8	0.5	✓	✓	DM priority	t	Figure 11(d)

• Deadline range d. The deadline of the task is randomly generated within an interval. This interval can be expressed as $[C_i + (1 - d) \times (T_i - C_i), T_i]$, where d specifies the deadline range. When $d = 0$, the interval of the deadline is $[T_i, T_i]$, which is the implicit-deadline sporadic task. When $d = 1$, the interval is $[C_i, T_i]$.

• Task release jitter J_i. It is generated in a uniformly distributed manner within the interval $[0, 0.05 \times Ti]$, which is tested from our actual RTOS.

• Task blocking time B_i. It is generated in a uniformly distributed manner within the interval $[0, C_{i+1}^{\max}]$, where $C_{i+1}^{\max}$ represents the maximum execution time among lower-priority tasks $\{\tau_{i+1}, \ldots, \tau_n\}$. Since the priority ceiling protocol [41] is used in a single processor, the maximum time that task τ_i is blocked by low-priority tasks is $C_{i+1}^{\max}$.

Each point yields 1000 task sets, and each experiment yields 20 points. Therefore, each experiment yields a total of 20000 task sets.

The experiment includes a comparison of several algorithms, namely Liu, Imp Lehoczky, Imp Hansson, RTA, RTA Optimal, Imp Davis, and Yu. These algorithms are described in Table 1 of the background and related work section. The selection of these algorithms is based on specific criteria: Liu is chosen for its fastest schedulability test, whereas other tests are the most advanced and accurate tests in their corresponding scenarios.

5.3 Experimental results

5.3.1 *Schedulable ratio*

Experiment 1. Comparison of the schedulable ratio when the utilization of the task set changes under random priority assignment. The experimental results are displayed in Figure 5(a).

Experiment 2. Comparison of the schedulable ratio when the utilization of the task set changes under DM priority assignment (i.e., the priority of the task is assigned according to deadline D_i). The experimental results are depicted in Figure 5(b).

Experiment 3. Comparison of the schedulable ratio when the utilization of the task set changes under RM priority assignment (i.e., the priority of the task is assigned according to period T_i). The experimental results are illustrated in Figure 5(c).

Result analysis. The task-assigned priorities according to the DM policy can achieve a high schedulable ratio. RTA is a classic exact schedulability test that provides necessary and sufficient conditions for determining schedulability. The RM is the optimal scheduling [13], and the upper bound of utilization in the Liu test is 0.69. For EBAI and other exact determination tests, the schedulable ratio drops rapidly when the utilization of the task set exceeds 0.8. Hence, the usage increases by 15.9% when using exact tests. Both EBAI and RTA are exact schedulability tests, resulting in the same schedulable ratio, which is higher than that of other sufficient determination tests.

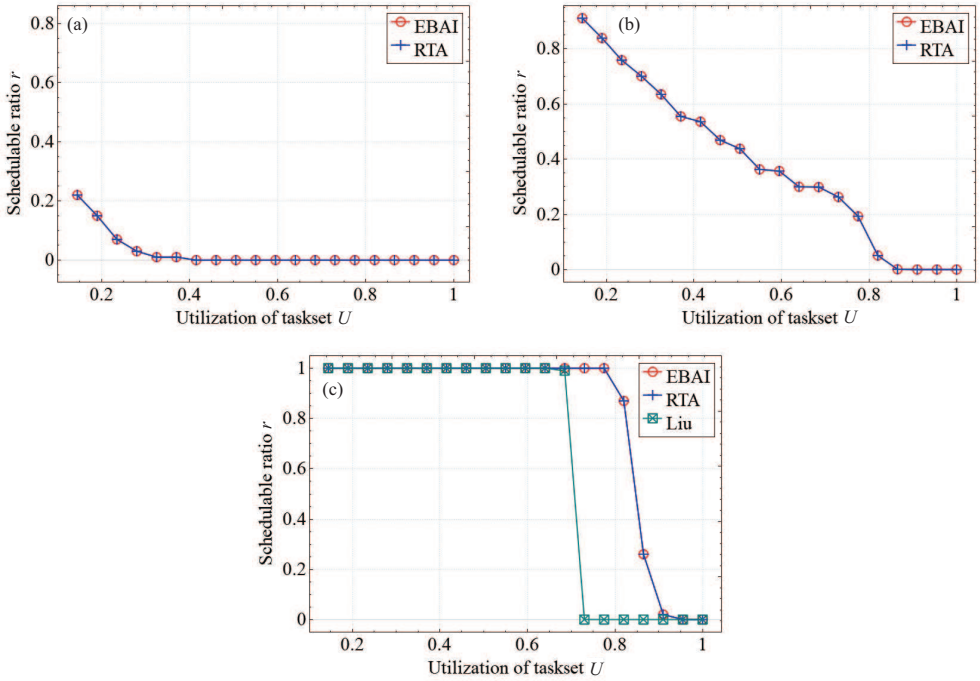

Figure 5 (Color online) Comparison of the schedulable ratio with the utilization of the task set. (a) Random priority assignment; (b) DM scheduling; (c) $J_i = 0$, $B_i = 0$, $D_i = T_i$, RM scheduling.

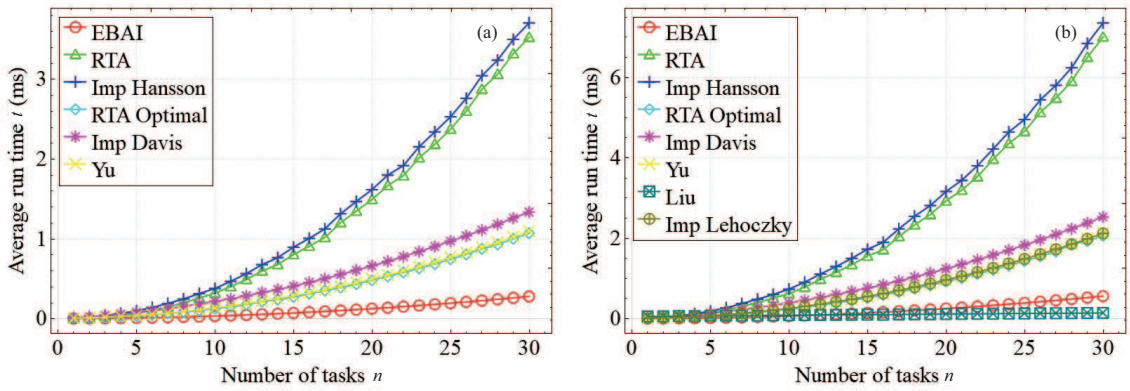

Figure 6 (Color online) Impact of the number of tasks on run time overhead. (a) DM scheduling; (b) $J_i = 0$, $B_i = 0$, $D_i = T_i$, RM scheduling.

5.3.2 *Runtime overhead*

Experiment 4. To enhance the schedulability ratio, tasks are assigned priority based on the DM policy. We conducted a comparison of the run time overhead as the number of tasks changed. As shown in Figure 6(a), all schedulability tests belong to exact schedulability tests. The Imp Hansson test shows the fastest increase in run time overhead, while the EBAI test shows the slowest increase, which is consistently lower than other tests. When the number of tasks is less than 5, the time cost of all tests is nearly equal. However, with 30 tasks, the RTA Optimal and Yu tests show the lowest time cost, approximately 1.15 ms. In contrast, the EBAI test is only 0.31 ms, which is reduced by 73%. As the number of tasks increases, the gap between the EBAI test and other tests becomes larger. Since the time slice of an actual embedded RTOS is generally 1 ms, the EBAI test can be completed within a single time slice. This improvement in real-time performance brings the EBAI test to a practical application level.

Experiment 5. Comparison of the run time overhead when the number of the tasks changes under RM priority assignment, as shown in Figure 6(b). Compared with Experiment 4, the run time overhead of all tests has increased by nearly double.

Experiment 6. Comparison of the run time overhead when the utilization of the task set changes

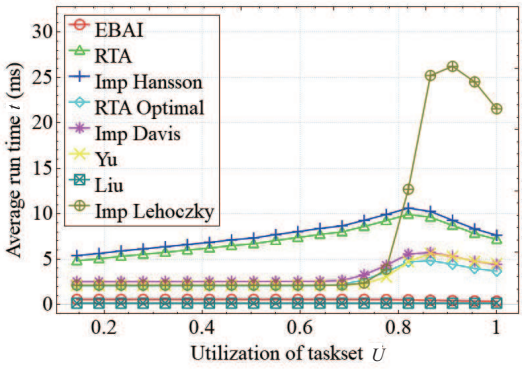

Figure 7 (Color online) Impact of task set utilization on run time overhead.

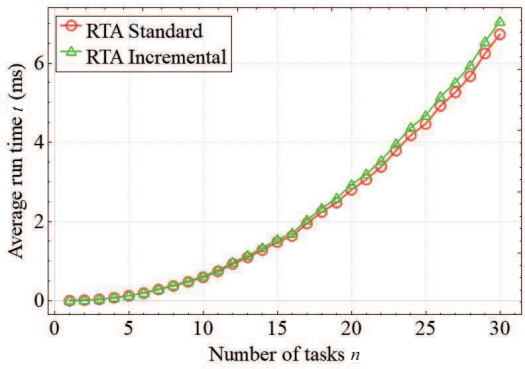

Figure 8 (Color online) Impact of task number on run time overhead.

under RM priority assignment, as shown in Figure 7.

Result analysis. For Figure 6(a), the run time overhead increases square [30] based on the response time analysis when the task parameters are integers and the maximum period is a finite value. The initial value $R_i^0 = R_i^{\mathrm{LB}} = (C_i + B_i + \sum_{j<i} J_j U_j)/(1 - \sum_{j<i} U_j)$ in Imp Hansson test is slightly larger than the initial value $R_i^0 = C_i + B_i$ in RTA, hence Imp Hansson test makes the number of iterations slightly reduced. However, this initial value requires floating-point operations. In embedded microprocessors that do not contain or disable floating-point units, the schedulability test can cause the total run time overhead to be slightly larger than the original initial value of RTA. The EBAI first quickly calculates $C_i + B_i + J_i + \sum_{j<i} \mathrm{WCIT}_{j,i}(D_i)$ and compares it with D_i, which performs a sufficient test. This approach can directly test the schedulability of the task to a large extent. If the test fails, the optimal exact initial value in (10) will be used for RTA to test again, resulting in a very low total run time overhead. For Figure 6(b), since Liu belongs to upper bound analysis tests, the time complexity is only $O(n)$. Nevertheless, EBAI is only slightly higher than the Liu test in the run time overhead. However, from Experiment 3, it can be seen that the schedulable ratio of EBAI is much higher than that of Liu. Due to the higher schedulability ratio observed in this experiment compared to Experiment 4 (we can know this from Experiments 2 and 3), the program cannot exit early. RTA may test schedulability from the first task to the last task in the task set, resulting in a higher run time overhead than in Experiment 4. For Figure 7, in the Imp Lehczky test, more time points need to be checked when the task is difficult to schedule, leading to a significant increase in run time overhead. Therefore, the run time overhead may fluctuate greatly when the task set reaches an unschedulable critical utilization.

5.3.3 *Conclusion about EBAI*

EBAI is an exact schedulability test that can determine the schedulability of tasks with release jitter and blocking time, providing a Boolean result. Moreover, it can start testing in an arbitrary task priority order. These characteristics make EBAI have strong versatility for schedulability determination. Additionally, when compared to other exact schedulability tests, EBAI shows the lowest run time overhead. When considering the release jitter and blocking time of tasks and using the DM scheduling with a task number of 30, the state-of-the-art test shows a time cost of 1.15 ms, whereas the EBAI test only requires 0.31 ms, resulting in a reduction of 73%. The efficiency gain of our test increases with the number of tasks. Compared with sufficient tests, the run time overhead of EBAI is slightly higher, but the schedulable ratio is much higher. The usage is increased by 15.9% than that of the usage upper bound tests. When it is a periodic task model and the utilization of the task set is higher than 0.69, the schedulability ratio of Liu's upper bound algorithm decreases to 0, while the schedulability ratio of exact schedulability tests decreases to 0 only when the utilization of task set is higher than 0.8. Therefore, the efficiency of the schedulability test is greatly improved by EBAI.

5.3.4 *Experiments about improved existing tests*

Experiment 7. Comparison of average run time overhead is performed between the RTA incremental implementation and the standard implementation (as shown in Algorithms 2 and 3). The average run time overhead comparison is shown in Figure 8.

Algorithm 2 RTA standard (S_H)

Require: Real-time task set S_H.
Ensure: The schedulability of S_H.

```
for τ_i : S_H do
    R_1 = initial_value();
    R_2 = 0;
    while 1 do
        Sum = 0;
        for j = 0; j < i; ++j do
            Sum+ = ceiling(R_1/T_j) × C_j;
        end for
        R_2 = C_i + Sum;
        if R_1 < R_2 then
            if R_2 > D_i then
                Return false;
            end if
            R_1 = R_2;
        else
            Break;
        end if
    end while
end for
Return true.
```

Algorithm 3 RTA incremental (S_H)

Require: Real-time task set S_H.
Ensure: The schedulability of S_H.

```
for τ_i : S_H do
    R_1 = initial_value();
    R_2 = R_1;
    for j = 0; j < i; ++j do
        inter[j] = ceiling(R_1/T_j) × C_j;
        R_2+ = inter[j];
    end for
    R_1 = R_2;
    while 1 do
        for j = 0; j < i; ++j do
            tmp = ceiling(R_1/T_j) × C_j;
            R_2+ = (tmp − inter[j]);
            inter[j] = tmp;
        end for
        if R_1 < R_2 then
            if R_2 > D_i then
                Return false;
            end if
            R_1 = R_2;
        else
            Break;
        end if
    end while
end for
Return true.
```

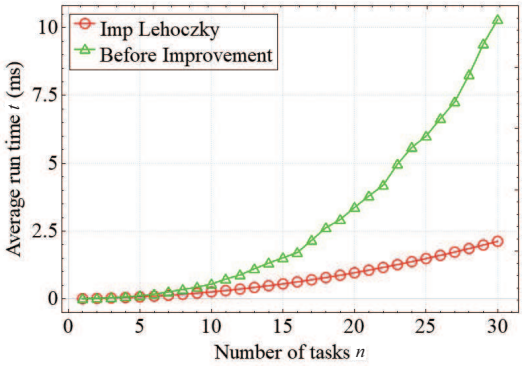

Figure 9 (Color online) Comparison of run time overhead between Imp Lehoczky and before improvement.

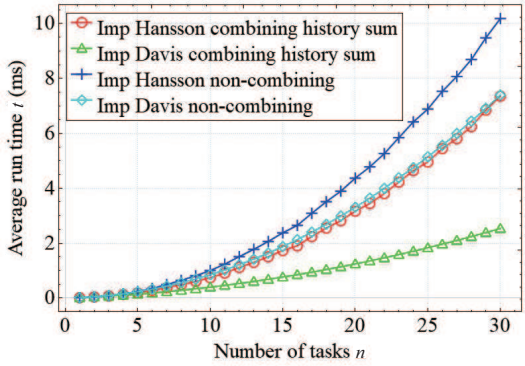

Figure 10 (Color online) Comparison of run time overhead between combining and non-combining historical sum.

Experiment 8. Comparison of the average run time overhead is performed between the Imp Lehoczky test and before improvement. As shown in Figure 9, it is obvious that the run time overhead is lower after improvement. Specifically, when the number of tasks is 30, the test before the improvement requires 10.3 ms, while the test after the improvement only requires 2.1 ms, making a reduction of 79.6%.

Experiment 9. Comparison of Imp Hansson and Imp Davis test between the way of combining the historical sum and the non-combined one is performed. As shown in Figure 10, It indicates that when the number of tasks is 30, the run time overhead of the Imp Hansson combining history sum is 7.5 ms, which is 28.6% lower than that of the uncombined historical sum test. Similarly, the run time cost of the Imp Davis test is 2.6 ms, which is lower than that of the uncombined with historical sum test, making a reduction of 65.3%.

5.3.5 *Experiments about WCIT*

Experiments 10–13. WCIT and EBAI were compared in terms of schedulable ratio and runtime overhead. WCIT is capable of conducting a sufficient test, whereas EBAI can perform an exact schedulability test. The experimental results of the comparisons of schedulable ratio and runtime overhead are

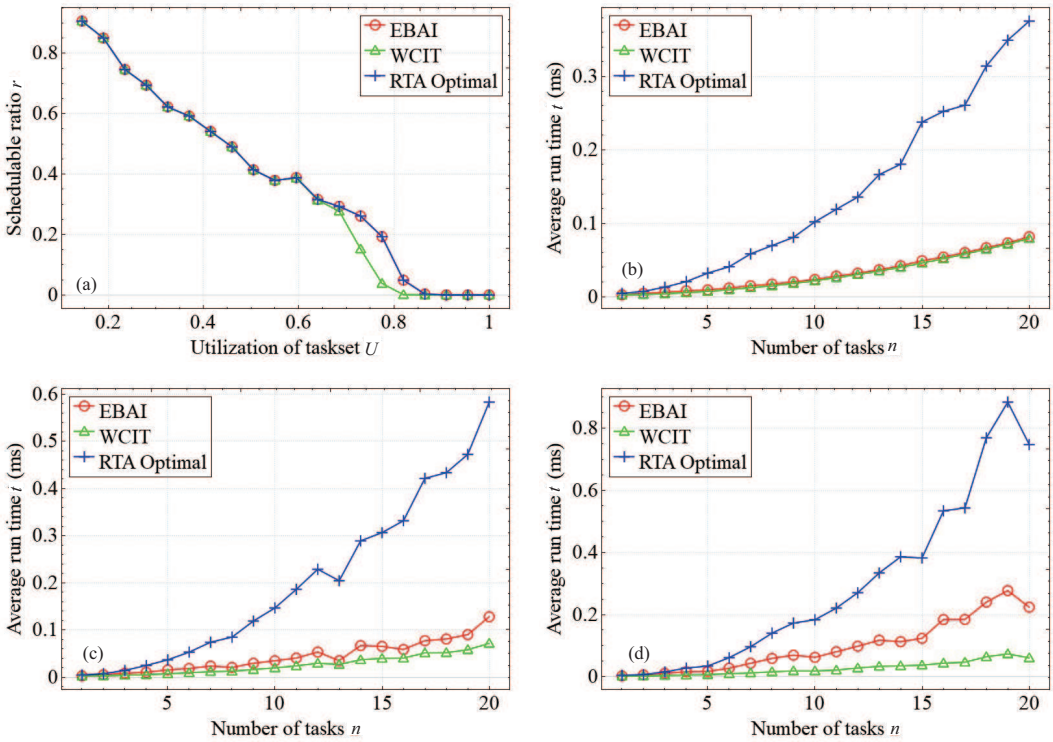

Figure 11 (Color online) (a) Comparison of the schedulable ratio of EBAI, WCIT and RTA optimal. Comparison of run time overhead of EBAI, WCIT and RTA optimal when utilization of task set is (b) 0.6, (c) 0.7, and (d) 0.8.

respectively depicted in Figure 11(a) (Experiment 10) and Figure 11(b) (Experiment 11), Figure 11(c) (Experiment 12), and Figure 11(d) (Experiment 13), where the utilization of the task set is 0.6, 0.7, 0.8, respectively.

Result analysis. It can be observed from Figure 11(a) that the difference in the schedulability ratio between WCIT and EBAI is relatively small. The schedulable ratio of WCIT is lower than that of EBAI only when the task set utilization is relatively high. This indicates that WCIT can accurately test the schedulability of tasks under low CPU load conditions, eliminating the need for further testing with RTA in EBAI and making the runtime overhead of WCIT and EBAI nearly equal. As the number of tasks increases, the schedulable ratio of the task set decreases. Therefore, EBAI needs to combine the RTA for additional testing, which is more time-consuming and results in the runtime overhead of EBAI being slightly higher than WCIT, as shown in Figure 11(b). Moreover, with the increase in task set utilization, the runtime overhead of EBAI is substantially higher than that of WCIT, as shown in Figures 11(c) and (d). This can be attributed to the increase in CPU load leading to a corresponding decrease in the schedulable ratio (as seen in Experiment 2), requiring more intervention from RTA to further test the schedulability. Although EBAI slightly increases the runtime compared to WCIT, it becomes an exact schedulability test and can further improve processor utilization. The value of this approach is evident, making it worthwhile and effective in implementing EBAI.

6 Conclusion

Aiming at the classic schedulability analysis problem in fixed-priority preemptive scheduling systems under a uniprocessor, we consider the release jitter and blocking time of tasks and determine the schedulability of task sets under arbitrary priority assignment. First, an exact Boolean schedulability test, EBAI, based on WCIT, is proposed. Second, the schedulability test efficiency of three existing algorithms is enhanced, and the related research results in this field over the years are summarized. Finally, EBAI is compared with all related results in several experiments. The experimental results show that EBAI is more efficient than the state-of-the-art test. In certain cases, the runtime gain of our test may exceed 73%. The EBAI is oriented to the five-tuple real-time task model with stronger expression ability and supports schedulability testing of arbitrary task priority order. Moreover, EBAI is an exact

schedulability test with the advantage of strong versatility and high efficiency, attributes that enable its implementation in practical applications. The spacecraft computer system is the most crucial embedded system. Therefore, our method can be easily applied to other embedded systems. These characteristics will enable its widespread use in industry, particularly in applications such as online admission control in an RTOS, swift delivery of schedulability results within an industry design tool, and rapidly assigning feasible priority to a control system. Subsequently, EBAI cannot provide the WCRT of tasks, which limits some application scenarios, such as non-preemptive scheduling systems, in which the WCRT is the main analysis parameter. Additionally, in future work, we will consider analysis methods in these specific scenarios and multiprocessor applications when release jitter and blocking time of the task are considered.

Acknowledgements This work was supported by National Natural Science Foundation of China (Grant Nos. 62032004, 61802017, 61632005).

References

1 Akesson B, Nasri M, Nelissen G, et al. An empirical survey-based study into industry practice in real-time systems. In: Proceedings of the Real-Time Systems Symposium, Houston, 2020. 3–11
2 Stankovic J A. Misconceptions about real-time computing: a serious problem for next-generation systems. Computer, 1988, 21: 10–19
3 Audsley N, Burns A, Davis R, et al. Real-time System Scheduling. Heidelberg: Springer, 1995. 41–52
4 Baruah S K, Cohen N K, Plaxton C G, et al. Proportionate progress: a notion of fairness in resource allocation. Algorithmica, 1996, 15: 600–625
5 Tang Y, Guan N, Feng Z, et al. Response time analysis of lazy round robin. In: Proceedings of the Design, Automation & Test in Europe Conference & Exhibition, Grenoble, 2021. 258–263
6 Ramamritham K, Stankovic J A. Scheduling algorithms and operating systems support for real-time systems. Proc IEEE, 1994, 82: 55–67
7 Jian G, Mengfei Y. Evolutionary fault tolerance method based on virtual reconfigurable circuit with neural network architecture. IEEE Trans Evol Computat, 2018, 22: 949–960
8 Li S, Qiao L, Yang M. Memory state verification based on inductive and deductive reasoning. IEEE Trans Rel, 2021, 70: 1026–1039
9 Wang S J, Xu W, Zheng X Y. Research on the technique of module dynamic loading for satellite software. In: Proceedings of the International Conference on Instrumentation, Measurement, Computer, Communication and Control, Shenyang, 2013. 813–816
10 Wang Y. Design and dynamic update of real-time systems. In: Proceedings of the Real-Time Systems Symposium, Hong Kong, 2019. 1–3
11 Ma Z, Qiao L, Yang M F, et al. Verification of real time operating system exception management based on SPARCv8. J Comput Sci Technol, 2021, 36: 1367–1387
12 Ekberg P. Rate-monotonic schedulability of implicit-deadline tasks is NP-hard beyond Liu and Layland's bound. In: Proceedings of the Real-Time Systems Symposium, Houston, 2020. 308–318
13 Liu C L, Layland J W. Scheduling algorithms for multiprogramming in a hard-real-time environment. J ACM, 1973, 20: 46–61
14 Bini E, Buttazzo G C, Buttazzo G M. Rate monotonic analysis: the hyperbolic bound. IEEE Trans Comput, 2003, 52: 933–942
15 Lauzac S, Melhem R, Mosse D. An improved rate-monotonic admission control and its applications. IEEE Trans Comput, 2003, 52: 337–350
16 Lehoczky J, Sha L, Ding Y. Rate monotonic scheduling algorithm: exact characterization and average case behavior. In: Proceedings of the Real-Time Systems Symposium, Santa Monica, 1989. 1–3
17 Sjodin M, Hansson H. Improved response-time analysis calculations. In: Proceedings of the Real-time Systems Symposium, Madrid, 1998. 399–408
18 Audsley N C. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. 1991. https://personal.utdallas.edu/~cxl137330/courses/fall13/RTS/papers/5.pdf
19 Leung J Y T, Whitehead J. On the complexity of fixed-priority scheduling of periodic, real-time tasks. Perform Eval, 1982, 2: 237–250
20 Liu H, Xi C, Qiao L, et al. A schedulability test for sporadic task DM scheduling based on density upper bound. IEEE Access, 2022, 10: 12475–12486
21 Abdelzaher T F, Sharma V, Lu C. A utilization bound for aperiodic tasks and priority driven scheduling. IEEE Trans Comput, 2004, 53: 334–350
22 Ekberg P, Yi W. Fixed-priority schedulability of sporadic tasks on uniprocessors is NP-hard. In: Proceedings of the Real-Time Systems Symposium, Paris, 2017. 139–146
23 Bonifaci V, Marchetti-Spaccamela A, Megow N, et al. Polynomial-time exact schedulability tests for harmonic real-time tasks. In: Proceedings of the Real-time Systems Symposium, Vancouver, 2013. 236–245
24 Chen J J, Huang W H, Liu C. k2U: a general framework from k-point effective schedulability analysis to utilizationbased tests. In: Proceedings of the Real-time Systems Symposium, San Antonio, 2015. 107–118
25 Baker T P. Stack-based scheduling of realtime processes. J Real-Time Syst, 1991, 3: 67–99
26 Tindell K. Using Offset Information to Analyse Static Priority Pre-emptively Scheduled Task Sets. York: University of York, Department of Computer Science, 1992
27 Joseph M, Pandya P. Finding response times in a real-time system. Comput J, 1986, 29: 390–395
28 Audsley N, Burns A, Richardson M, et al. Applying new scheduling theory to static priority pre-emptive scheduling. Softw Eng J UK, 2002, 8: 284–292
29 Eisenbrand F. Static-priority real-time scheduling: response time computation is NP-hard. In: Proceedings of the Real-time Systems Symposium, Barcelona, 2008. 397–406
30 Baruah S, Bertogna M, Buttazzo G. Multiprocessor Scheduling for Real-time Systems. Berlin: Springer, 2015

31 Bril R J, Verhaegh W, Pol E. Initial values for online response time calculations. In: Proceedings of the 15th Euromicro Conference on Real-Time Systems, Porto, 2003. 13–22

32 Lu W C, Lin K J, Wei H W, et al. Period-dependent initial values for exact schedulability test of rate monotonic systems. In: Proceedings of International Parallel and Distributed Processing Symposium, Long Beach, 2007. 26–30

33 Davis R I, Zabos A, Burns A. Efficient exact schedulability tests for fixed priority real-time systems. IEEE Trans Comput, 2008, 57: 1261–1276

34 Bini E, Baruah S K. Efficient computation of response time bounds under fixed-priority scheduling. In: Proceedings of the 15th Conference on Real-Time and Network Systems, 2007

35 Davis R I, Burns A. Response time upper bounds for fixed priority real-time systems. In: Proceedings of the Real-time Systems Symposium, Barcelona, 2008. 407–418

36 Yu G L, Yang M F, Sun H J, et al. Improving the efficiency of schedulability tests for fixed priority preemptive systems. In: Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing. Berlin: Springer, 2016. 31–44

37 Günzel M, Teper H, Chen K H. Work-in-progress: evaluation framework for selfsuspending schedulability tests. In: Proceedings of the Real-time Systems Symposium, Dortmund, 2021. 532–535

38 Yadlapalli Y, Liu C. Lag-based analysis techniques for scheduling multiprocessor hard real-time sporadic dags. In: Proceedings of the Real-Time Systems Symposium, Dortmund, 2021. 316–328

39 Liu H B, Qiao L, Yang M F, et al. Real-time task scheduling and analysis method based on virtual zoom out period (in Chinese). J Softw, 2022, 33: 1–17

40 Dong Z, Yang K, Fisher N, et al. Tardiness bounds for sporadic gang tasks under preemptive global EDF scheduling. IEEE Trans Parallel Distrib Syst, 2021, 32: 2867–2879

41 Sha L, Rajkumar R, Lehoczky J P. Priority inheritance protocols: an approach to real-time synchronization. IEEE Trans Comput, 1990, 39: 1175–1185

42 Bini E, Buttazzo G C. Measuring the performance of schedulability tests. Real-Time Syst, 2005, 30: 129–154