Abstract

Time demand analysis of real-time has been an active research area and a plethora of results have been derived for the classing time demand analysis by relaxing the assumption such as making task periods harmonic, restricting the scheduling points to a subset, obtain guess values, and avoiding testing system feasibly at unnecessary points. However, till date, the complexity of the time demand analysis still remains pseudo-polynomial. In this work, we propose a higher initial value for testing the feasibility of a lower priority task based on the feasibility analysis of immediate higher priority task. The proposed technique shows improvement over closely related counterparts. Experimental results are aligned with theoretical formulations presented in this paper.

Keywords: Operating Systems, Time Demand Analysis Real-Time Systems, Fixed-Priority Scheduling, Feasibility Analysis, Online Schedulability Test.

INTRODUCTION

A major challenge in the design of real-time embedded system is to validate its correctness. Such systems are not only expected to be logically correct but the timing constraints of these systems must also be respected. Various scheduling techniques have been proposed in literature to verify the correctness of real-time system. The real-time scheduling algorithms can be divided into two main types i.e., preemptive and non-preemptive systems. Though non-preemptive are simple when it comes to implementation, such policy loses its correctness of real-time system. The real-time scheduling system, there exist a number of feasibility tests [1, 4-10]. These tests can be partitioned into two broad categories i.e., Necessary and Sufficient Conditions (NSC), and Sufficient Conditions (SC). On one hand, NSC results in higher system utilization and can schedule real-time tasks in a system as long as the utilization is not more than 100%, while SC can promise system utilization up to the ln(2) only, where “n” is the number of tasks in the system. However, the complexity of NSC is pseudo-polynomial and restricts its use in online systems. On the other hand, SC are simple with O(n) complexity. Literature shows reveals that the lower complexity of SC comes at the price of lower system utilization while the complexity of NSC class is NP-hard in strong sense [11]. Consequently, variants of NSC have been proposed recently to lower the computational cost of feasibility tests instead of time complexity [3-10].

The problem of scheduling fixed systems was first addressed by Liu and Layland in 1973 [1] under simplified assumptions and authors derived the optimal static priority scheduling algorithm called rate monotonic (RM) algorithm for implicit-deadline model (when deadlines coincide with respective periods). Since then, to test the schedulability of fixed priority scheduling system, there exist a number of feasibility tests [1, 4-10]. These tests can be partitioned into two broad categories i.e., Necessary and Sufficient Conditions (NSC), and Sufficient Conditions (SC). On one hand, NSC results in higher system utilization and can schedule real-time tasks in a system as long as the utilization is not more than 100%, while SC can promise system utilization up to the ln(2) only, where “n” is the number of tasks in the system. However, the complexity of NSC is pseudo-polynomial and restricts its use in online systems. On the other hand, SC are simple with O(n) complexity. Literature shows reveals that the lower complexity of SC comes at the price of lower system utilization while the complexity of NSC class is NP-hard in strong sense [11]. Consequently, variants of NSC have been proposed recently to lower the computational cost of feasibility tests instead of time complexity [3-10].

The preemptive class of scheduling algorithms can be further classified into two major domains: (i) fixed priority, and (ii) dynamic priority [1, 2]. The main difference between both types is the priority assignment. Under fixed-priority algorithm, each task is assigned a unique priority that remains fixed as long the task set is under operation. In the dynamic priority techniques on the other hand, the priority of the task may change at run time and hence becomes unpredictable when the system becomes overloaded. In addition, fixed priority systems are simple from implementation perspective and can be easily implemented atop many available operating systems.

The remaining paper is organized as follows. Section 2 covers the background work and constructs the model to formulate our problem. The technique for obtaining higher initial value
is described in Section 3, while Section 4 discusses experimental results. The paper is concluded in Section 5.

BACKGROUND AND PROBLEM FORMULATIONS

Before we discuss the background work, we introduce the task model that is used for deriving our main results in Section 3.

Let \( \tau = \{\tau_1, ..., \tau_n\} \) be a periodic task system, where each task \( \tau_i \) is represented by its parameters execution time \( c_i \), task period \( p_i \), and deadline \( d_i \). To formulate our model, we assume that the tasks are independent of each other and there is only a single processor available to schedule all the tasks in the system using RM. Being a fixed priority system, RM assigns static priorities on task activation rates (periods). For constrained deadline systems, when periods are larger than deadlines, an optimal priority technique was drawn in [13] called Deadline-Monotonic (DM) system. The RM and DM are identical when relative deadline of every task is proportional to its period. For simplicity, we assume implicit deadline model i.e., \( p_i = d_i \). The utilization of task \( \tau_i \) is defined as: \( u_i = c_i / p_i \). The cumulative utilization \( u_{\text{tot}} \) of periodic task system \( \tau \) is:

\[
\[1\] \quad u_{\text{tot}} = \sum_{i=1}^{n} \frac{c_i}{p_i}
\]

For validating timing constrains, feasibility tests—given a task set and system model, determining whether it is possible to meet all the deadlines—are performed to achieve system predictability [4-7, 14, 16]. The first feasibility test for RM was proposed in 1973 [6], by Liu and Leyland. According to [6], a periodic task system is schedulable if

\[
\[2\] \quad u_{\text{tot}} \leq n(2^{1/n} - 1)
\]

Where \( n \) denotes the number of tasks in \( \tau \).

Equation 2 can only promise the feasibility of the system as long its utilizations less than 69% [1]. To overcome the theoretical difference in performance proposed by LL-bound, necessary and sufficient condition (NSC) based tests were proposed [1-9]. The feasibility can be either straightforward approaches [5-7] or iterative [8-9]. The straightforward solution test task feasibility only at times when tasks arrive, called scheduling points. The iterative techniques test task feasibility by employing iteration. Under both implementation, the time complexity remains pseudo-polynomial [11-12] and hence the focus is on reducing the computation cost of these techniques.

The workload due to \( \tau_i \) at time \( t \) consists of its execution demand \( c_i \) as well as the interference it encounters due to higher priority tasks from \( \tau_{i-1} \) to \( \tau_1 \) and can be expressed mathematically as:

\[
\[3\] \quad W_i(t) = c_i + \sum_{j=1}^{i-1} \left( \left\lfloor \frac{t}{p_j} \right\rfloor c_j \right)
\]

A periodic task \( \tau_i \) is feasible if we find some \( t \in [0, t] \) satisfying

\[
\[4\] \quad L_i = \min_{0 \leq t \leq p_i} W_i(t) \leq t
\]

In other words, task \( \tau_i \) completes its computation requirements at time \( t \in [0, t] \), if and only if the entire request from the \( i - 1 \) higher priority tasks and computation time of \( \tau_i \) is completed at or before time \( t \). As \( t \) is a continuous variable, there are infinite numbers of points to be tested. The entire task set \( \tau \) is feasible iff

\[
\[5\] \quad L = \max_{i \in S_{\tau}} \min_{0 \leq t \leq p_i} W_i(t) / t \leq 1
\]

The first attempt to limit the infinite number of points in interval \( t \in [0, t] \) is made in [8]. The authors’ show that \( W_i(t) \) is constant, except at finite number of points, where tasks are released, called RM scheduling points. Consequently, to determine whether \( \tau_i \) is schedulable, \( W_i(t) \) is computed only at multiples of \( \tau_i \), \( 1 \leq j \leq i \).

Specifically, let

\[
\[6\] \quad S_i = \left\{ a p_b \mid b = 1, ..., i; a = 1, ..., \left\lfloor \frac{p_i}{p_b} \right\rfloor \right\}
\]

Under TDA, the fundamental theorem to determine whether an individual task is feasible or not.

**Theorem 1.**—Given a set of \( n \) periodic tasks \( \tau_1, ..., \tau_n \), \( \tau_i \) can be feasibly scheduled for all tasks phasings using RM iff

\[
\[7\] \quad L_i = \min_{i \in S_{\tau}} W_i(t) / t \leq 1
\]

Theorem 1 is known as TDA [7]. To reduce the computation cost associated with TDA, authors in [4] proposed hyperplanes test. The Hyperplanes Exact Test (HET) reduces scheduling point for \( \tau_i \) from set \( S_{\tau} \) to a reduced set \( H_{\tau_i}(t) \). For any task \( \tau_i \), their test begins with \( p_i \) and expands its search space by

\[
\[8\] \quad H_{\tau_i}(t) = H_{\tau_{i-1}} \left( \frac{t}{p_i} \right) \cup H_{\tau_{i-1}}(t)
\]

where \( H_0(t) = \{1\}. \)
Furthermore, Theorem 1 was also extended in [15] by deriving a technique called Enhanced Time demand Analysis (ETDA).

**Theorem 2.** Given a set of \( n \) periodic tasks \( \{ \tau_1, \ldots, \tau_n \} \), \( \tau_i \) can be feasibly scheduled for all tasks phases using RM iff

\[
L_i = \min_{t \in Z_i} \frac{W_i(t)}{t} \leq 1
\]  

(9)

where \( Z_i = S_j - X_{i-1} \) and \( X_{i-1} \) is the set of scheduling points at which the schedulability of \( T_{i-1} \) is negative. By extension \( X_0 = \emptyset \).

It is evident from Inequality 9 that the TDA has been improved from computational perspective by restricting the scheduling points. Similarly, we extend the TDA by obtaining a higher initial value for any low priority task that help in avoiding many unnecessary steps.

**HIGHER INITIAL VALUES FOR TASKS**

In our approach, the system feasibility is tested in the highest priority first fashion where the test proceeds to the lower priority task only if the current periodic task is RM schedulable; otherwise the system is infeasible as per RM scheduling. With Theorem 1, the starting guess for a task \( \tau_i \) is \( c_i \) and hence feasibility analysis starts with first scheduling point in the set of scheduling points \( S_i \), i.e., \( p_i \in S_i \); \( \min_{t \in S_i}(W_i(t)/t) \leq 1 \). With this formulation, it is evident for a lower priority task \( \tau_{i+1} \) that its workload \( W_{i+1}(t) \) is again tested at \( p_i \) while the demand at point \( p_i \) is now higher than what was presented by \( \tau_i \) at same point \( p_i \). However, it cannot be concluded that the same \( p_i \) that accommodates the workload of \( \tau_i \) can also handle the workload due to \( \tau_{i+1} \) as \( c_{i+1} \) is additional term contributing to the workload at \( p_i \) by \( \tau_{i+1} \). This pattern suggests that \( \tau_{i+1} \) has to be tested at \( p_i \) and so on, unless the workload becomes less than or equal to the available time on a single processor system.

Let \( W_{i,j}(t) \) is the first value for \( \tau_j \) and \( W_{i-1,j}(t) \) is the workload due to cumulative workload of lower priority tasks \( \{ \tau_1, \tau_2, \ldots, \tau_{i-1} \} \) which is feasibly at \( t \). For \( W_{i-1,j}(t) \), \( t \) is the first point \( t \in S_j \) at which the schedulability of \( T_{j-1} \) is answered. Since \( \tau_j \) is unschedulable at \( t \) and hence \( W_{j}(t) > t \). Therefore, schedulability test now skips the remaining point in set as condition i.e., \( \min_{t \in S_j} \) is true.

We now explain the working of our solution in Table 1 which highlights the feasibility analysis of a task where the task computation and period may have random values such that computation demand of a task is not more than its respective deadline. Consider, Table 1 is being populated while analyzing the workload at multiple of higher tasks time periods, starting with highest priority first analysis approach, for a task set consisting of four tasks. As shown, the last value for any task \( \tau_i \) becomes the first candidate value for the task \( \tau_{i+1} \) at which its feasibility has to be tested. For task \( \tau_2 \) in step\#4, \( W_2(t) \) is satisfied over point \( t_5 \) and hence \( W_2(t) \) is declared \( w_2(t) \). Consequently the starting value for \( \tau_i \) becomes \( w_3(t) = w_2(t) \).

<table>
<thead>
<tr>
<th>Task#</th>
<th>Step#</th>
<th>( S_j )</th>
<th>Testing Point</th>
<th>Workload</th>
<th>( L_i ) ≤ 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>( t_1 )</td>
<td>( t_1 \in S_1 )</td>
<td>( w_1(t_1) )</td>
<td>✓</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>( t_1, t_2, t_3 )</td>
<td>( t_4 \in S_2 )</td>
<td>( w_2(t_4) )</td>
<td>×</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>( t_1, t_2, t_3 )</td>
<td>( t_4 \in S_2 )</td>
<td>( w_2(t_4) )</td>
<td>×</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>( t_1, t_2, t_3 )</td>
<td>( t_4 \in S_2 )</td>
<td>( w_2(t_4) )</td>
<td>✓</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>( t_1, t_2, t_3, t_4 )</td>
<td>( t_4 \in S_3 )</td>
<td>( w_3(t_4) )</td>
<td>×</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>( t_1, t_2, t_3, t_4 )</td>
<td>( t_5 \in S_3 )</td>
<td>( w_3(t_5) )</td>
<td>×</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>( t_1, t_2, t_3, t_4 )</td>
<td>( t_5 \in S_3 )</td>
<td>( w_3(t_5) )</td>
<td>×</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>( t_1, t_2, t_3, t_4 )</td>
<td>( t_5 \in S_3 )</td>
<td>( w_3(t_5) )</td>
<td>✓</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>( t_1, t_2, t_3, t_4, t_5 )</td>
<td>( t_4 \in S_4 )</td>
<td>( w_4(t_4) )</td>
<td>×</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>( t_1, t_2, t_3, t_4, t_5 )</td>
<td>( t_5 \in S_4 )</td>
<td>( w_4(t_5) )</td>
<td>×</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>( t_1, t_2, t_3, t_4, t_5 )</td>
<td>( t_5 \in S_4 )</td>
<td>( w_4(t_5) )</td>
<td>×</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>( t_1, t_2, t_3, t_4, t_5 )</td>
<td>( t_5 \in S_4 )</td>
<td>( w_4(t_5) )</td>
<td>×</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>( t_1, t_2, t_3, t_4, t_5 )</td>
<td>( t_5 \in S_4 )</td>
<td>( w_4(t_5) )</td>
<td>✓</td>
</tr>
</tbody>
</table>
computation associated with $\tau_{i+1}$. The workload that is satisfied at $t$ for $\tau_i$ is $w_i(t)$, so for $\tau_{i+1}$, it becomes the initial value. With same argument the workload at any point $t$ is:

$$w_i(t) < w_{i+1}(t) < \ldots < w_n(t)$$

and hence:

$$w_i'(t) < w_{i+1}'(t) < \ldots < w_n'(t) \quad (10)$$

We now apply Inequality 10 to the task set provided in Table 1. Our formulation provides the initial values for task $\tau_2$ in step#4 and hence feasibility test skips step#5-7. For the fourth task $\tau_4$, it skips 4 points more and so on. The advantage of our test becomes more visible when applied to larger tasks sets as higher initial values are obtained for the lower priority tasks and hence the test feasibility is determined much faster. We represent this scheme by Higher Time Demand Analysis (HTDA).

Figure 1 provides the graphical presentation of HTDA. We plot the time demand of tasks and identify the scheduling point that accommodates the workload presented by a task $T_i$. The x-axis represents the time while y-axis shows the demand of tasks at a given point in time. The region below the line having slope 1 is feasible for any task according to RM our approach. The dotted lines represent the workload of individual tasks that are non-decreasing and monotonically increase at task periods of all higher priority tasks. It can be seen in Figure 1 that the jump between $w_i(t)$ and $w_{i+1}(t)$ at any point in time $t$ is least $c_{i+1}$ units. The think dots on x-axis represent the time where the workload changes and cross highlights the first feasible scheduling point for a task where workload is satisfied at the given point. The feasibility of task $\tau_i$ becomes true at point reflected with cross. The shaded region identifies the values that are unnecessary for lower priority task $\tau_{i+1}$ and hence should ignore as they are obviously not going to address the demand. This shaded region constitutes the initial value for the next lower priority task and larger is this region, the better it would be for the lower priority task. For testing the feasibility of lower priority task $\tau_{i+1}$, a higher value is assigned to $\tau_{i+1}$ as initial cumulative demand. This is presented by $w_i'$ and $w_{i+1}'$ for task $\tau_i$ and $\tau_{i+1}$, respectively.

**EXPERIMENTAL RESULTS AND ANALYSIS**

To align with previous techniques presented in literature, we evaluate the performance of HTDA and compare the results with ETDA and HET from run time perspective. We generate random task periods from 10-100 tasks with step size of 5 tasks. In our task set generation module, no tasks have the same task period and periods are in the range of [10, 1000] with uniform distribution. Random values are taken for corresponding task execution demands within $[1, p_i]$. For
task computation values, we use uniform distribution. The priority of individual task has been assigned as per RM assignment criteria. Experimentation is done in MATLAB and running on a PC with 3.40GHz Intel (i7-3770) and 8GB RAM under Linux. We only analyze the run time performance as a rule of thumb, as the time taken to solve feasibility problem is the simple criteria for evaluating the performance of a given algorithm.

The performance of all techniques is better for the task sets when the number of task is low and increases as the size increases. When system utilization is 80% then its very likely that the cumulative demand is fulfilled with testing a few scheduling points and that shown in Figure 2(a). Even under 80% utilization, the increased number of task present more load for the test. Since ETDA needs to maintain a list of previously tested scheduling points and hence slower as compared to HET or HDTA. Similarly, the HET perform union operation while testing feasibility using recursive approach and hence the commutation cost is more than HTDA but lower than ETDA. Due to this recursive nature, HET is behaving similar to the ETDA when system utilization is 85% or 90%, as this is the utilization at which the feasibility has to be test maximum tasks and its very likely that the lowest priority task is also schedulable per RM algorithm.

Irrespective of utilization, HET has to confine the search space to a set number of scheduling points while HTDA just proceed with higher initial value. This trend is shown in Figure 2(a) to (d). Even when system utilization is 95%, HTDA outclass existing techniques due to its straight forward approach and under such utilization; it is very likely some tasks can miss the deadline. For lower utilization and less number of task, HTDA is very efficient as only points are analyzed while testing feasibility of a task. The worst case scenario is Figure 2(d) when system utilization is 95% and even for less number of task, nor scheduling point have to be analyzed which is aligned to our formulation.

![Figure 2: Performance analysis under varying system utilization](image-url)
CONCLUSION

The problem of analyzing the feasibility of periodic task under RM scheduling algorithm is discussed and a pattern between the two consecutive tasks schedulability is identified. The relationship between two neighboring tasks showed that any scheduling point which satisfies the CPU demand of a task becomes the initial value for the lower priority task at which the feasibility can be tested. The aforementioned relationship was exploited for faster feasibility analysis of time demand analysis for RM schedulability. Experimental results confirmed that the proposed method is quite competitive as compared to existing counterparts.

REFERENCES