Enhanced Time Demand Analysis

Nasro Min Allah, S. Islam and Wang Yong Ji

COMSATS Institute of Information Technology, Park Road, Islamabad, Pakistan
Laboratory for Internet Software Technologies, Institute of Software, Chinese Academy of Sciences, Beijing 100080, China

Abstract: Since its introduction, Rate Monotonic Analysis (RMA) has frequently been promoted as a way of attaining real-time system predictability, yet RMA (in its exact form), exhibits pseudo-polynomial time complexity and becomes impractical for larger task set to be analyzed online. To tune-up RMA for online systems, researchers are attempting to lower the complexity of RMA in terms of reducing the number of scheduling points. However, reducing number of scheduling points does not actually lower the number of points, which will be tested in actual, at run time. In this paper, we tackle the high complexity associated with current algorithms by identifying and propagating false points that arise during system feasibility analysis. The contribution of this paper is two-fold; first, it highlights an implicit drawback found in a recently proposed technique and secondly, a novel exact feasibility test called enhanced time demand analysis is presented, which significantly reduces the number of inequalities that would be tested otherwise.

Key words: Real-Time Systems · Fixed-priority Scheduling · Feasibility Analysis · Online Schedulability

INTRODUCTION

A major issue in the design of real-time multitasking systems is that of timing correctness i.e. they must provide predictable behavior under all circumstances. The solution lies in the optimal scheduling algorithm that assigns priorities to task on the basis of some predefined criteria such as activation rate or deadline etc. The most commonly used approach to schedule real time tasks is priority driven which falls into two types: fixed priority and dynamic priority [1]. A fixed-priority algorithm assigns the fixed/same priority to all jobs in each task, which should be different from the priorities assigned to jobs generated by other tasks in the system. In contrast, dynamic-priority scheduling algorithms place no restrictions upon the manner in which priorities are assigned to individual jobs. Although, dynamic algorithms are considered better theoretically [2], they become unpredictable when transient overload occurs [3]. This paper thus considers only fixed-priority scheduling due to its applicability, reliability and simplicity [2, 4-5].

The problem of scheduling periodic task under fixed-priority scheme was first addressed by Liu and Layland [6] in 1973 under simplified assumptions (for details see [6]), they derived the optimal static priority scheduling algorithm for implicit-deadline model (when deadlines coincide with respective periods) called rate monotonic (RM) algorithm. RM assigns static priorities on task activation rates (periods) such that for any two tasks and , priority ( ) < priority ( ) → period ( ) < period ( ) while ties are broken arbitrarily. For constrained deadline systems, where deadlines are not greater than periods, an optimal priority ordering has been shown in [7] to be Deadline-Monotonic (DM) scheduling, where priority assigned are inversely proportional to relative deadlines. The RM and DM are identical when relative deadline of every task is proportional to its period.

Let denotes a periodic task system. A task is represented by its parameters, , and , where is the execution time, is the time period and is the deadline of . For each task , its utilization is defined as: . We define a cumulative utilization of periodic task system as:

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

(1)

Corresponding Author: Nasro Min Allah, COMSATS Institute of Information Technology, Park Road, Islamabad, Pakistan

* A portion of this work, titled Revisiting Fixed Priority Techniques, is published in the "Embedded and Ubiquitous Computing (EUC 2007)" by Springer's LNCS 4808, pp. 134-145.
For validating timing constraints, 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 [4-5, 7-19]. The first feasibility test for RM was proposed by the same authors in [6], called LL-bound; a periodic task system is static-priority feasible if

\[ u_{\text{nc}} \leq n(2^n - 1) \]  

Where \( n \) denotes the number of tasks in \( \tau \). The term \( n(2^n - 1) \), decreases monotonically from 0.83 when \( n = 2 \) to \( \ln(2) \) as \( n \to \infty \). This shows that any periodic task set of any size is static priority feasible upon a preemptive uniprocessor, if the RM scheduling is used and \( u_{\text{nc}} \) is not greater than 0.693. This result gives a simple \( O(n) \) procedure to test task feasibility online, where tasks can arrive at run time, however, it is a sufficient condition only; it is quite possible that an implicit-deadline synchronous periodic task system which exceeding the LL-bound be static-priority feasible. The classic work of Liu and Layland [6] is extended in [1, 13-15], however all these tests are sufficient conditions (SC) only and trade utilization for performance.

To overcome the 3% theoretical difference in performance proposed by LL-bound, necessary and sufficient condition (NSC) based tests were proposed in literature [8-10, 12, 15-16]. These feasibility tests can be divided into two schemes: straightforward approaches [4-5, 8, 16] and iterative techniques [9, 10]. The former analyzes task feasibility only at times when tasks arrive, called scheduling points, while the later answers the task feasibility by employing iterative technique. Whatever is the implementation mechanism, their time complexity remains pseudo-polynomial [19]. Recently, authors in [4] extended the work in [8] by proposing an exact feasibility test that reduces the number of scheduling points. However, we show in Section 3 that reducing the number of scheduling point does not necessarily mean lowering the number of points which are actually being utilized by the test.

Unlike the previous work [4], this paper also extends the work in [8] and presents a novel technique to handle the complexity of exact test by reducing the number of points in actual that need to be tested otherwise for determining system feasibility. Though the main idea of this paper can be easily integrated into any fixed priority-scheduling algorithm, however in order to align with previous results, we focus on RM here. Simulation results show that our technique shows better performance, when compared to other techniques lying in the same category, without any modification to the underlying scheduling. As a related work, we report that recursive technique such as the one proposed in [4], involves recursive function calling and, in some cases even needs more points for analyzing feasibility of a task (details in Section 3).

The rest of paper is organized as follows. In Section 2, we introduce some basic notations to formulate our problem. The exact test [8], which provides the basis of our work, is presented in Section 3, in addition to the critical evaluation of hyperplane exact test [4]. The main result of this paper is presented in Section 4, where we derive a novel algorithm to determine the feasibility of task set at run time. Experimental results are discussed in Section 5 and finally, conclusion is drawn in Section 6.

**Problem Formulation and Notation:** Our work is targeting independent, preemptable and hard periodic task in which the response of the tasks may be earlier than the ends of their corresponding periods. In our model of a hard real-time task set, each task \( \tau_i \) generates a job at each integer multiple of \( p_i \) and each such job has an execution requirement of \( c_i \) time units that must be completed by the next integer multiple of \( p_i \). Moreover, all tasks immediately get ready for execution on uniprocessor as soon as they are released, deadlines are equal to periods, all tasks overheads such as task swapping times are ignored and the number of priority levels is unlimited. Initially all tasks arrives simultaneously at \( t = 0 \). To begin with, we describe a pseudo-polynomial time exact schedulability test in Section 3 for the feasibility of fixed priority systems, where the response time of the jobs are smaller than or equal to their respective periods. To determine whether a task can meet all its deadlines, we compute the total demand for processor time by a task \( \tau_i \), and check whether this demand can be met by its deadline.

Before we describe exact test in Section 3, we introduce some notations here, which will be needed latter in this paper. The workload constituted by \( \tau_i \) 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 \( \tau_i \) and can be expressed mathematically as

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

(3)

A periodic task \( \tau_i \) is feasible if we find some \( t \in [0, t] \) satisfying...
Table 1: RM scheduling points for 3-tasks set

<table>
<thead>
<tr>
<th>τ</th>
<th>c_i</th>
<th>p_i</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>c_1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>c_2</td>
<td>7</td>
<td>3.67</td>
</tr>
<tr>
<td>3</td>
<td>c_3</td>
<td>20</td>
<td>3.67, 7, 9, 12, 14, 15, 18, 20</td>
</tr>
</tbody>
</table>

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

In other words, task \( τ_i \) completes its computation requirements at time \( t \in [0, p_i] \), if and only if the entire request from the \( i-1 \) higher priority tasks and computation time of \( τ_i \), 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 if

\[ L = \max \left( \min_{0 \leq t \leq p_i} \frac{W_i(t)}{t} \right) \leq 1 \] (5)

Previous Results on Exact Tests

Time Demand Analysis: The first attempt to limit the infinite number of points in interval \( t \in [0, p_i] \) 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 \( \tau_i \leq t \), \( 1 \leq j \leq i \). Specifically, let

\[ S_i \{ \{a \rho_a | b = 1 \ldots \tau_i, \alpha = 1 \ldots \lfloor p_i / p_a \rfloor \} \} \] (6)

We have the following fundamental theorem to determine whether an individual task is feasible or not.

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

\[ L_i = \min_{0 \leq t \leq p_i} \frac{W_i(t)}{t} \leq 1 \] (7)

With above theorem, \( L_i \) is needed to be analyzed only at a finite number of points. The time complexity of Theorem 1 depends on both the number of tasks and maximum task period i.e. \( O(n \tau_i) \) [1]. In rest of the paper, we represent this technique by Time Demand Analysis (TDA). We now apply results parented in Theorem 1 to a set of three tasks with parameters given in Table 1.

In Algebraic Terms, We Have:

List 1
1. Task \( \tau_1 \) is RM-schedulable iff

\[ c_1 \leq 3 \] (8)

II. Task \( \tau_2 \) is RM-schedulable iff

\[ c_2 + c_3 \leq 3 \ OR \]
\[ 2 c_1 + c_3 \leq 6 \ OR \]
\[ 3 c_1 + c_2 \leq 7 \] (9)

III. Task \( \tau_3 \) is RM-schedulable iff

\[ c_1 + c_2 + c_3 \leq 3 \ OR \]
\[ 2 c_1 + c_1 + c_1 \leq 6 \ OR \]
\[ 3 c_1 + c_2 + c_2 \leq 7 \] (10)
\[ 3 c_1 + 2 c_2 + c_1 \leq 9 \ OR \]
\[ 4 c_1 + 2 c_2 + c_3 \leq 12 \ OR \]
\[ 4 c_1 + 2 c_2 + c_3 \leq 14 \ OR \]
\[ 5 c_1 + 3 c_2 + c_3 \leq 15 \ OR \]
\[ 6 c_1 + 3 c_2 + c_3 \leq 18 \ OR \]
\[ 7 c_1 + 4 c_2 + c_3 \leq 20 \ OR \]

For any task \( \tau_n \), the number of elements in set \( S_i \) is of particular interest. Every element of \( S_i \) contributes a RM scheduling point and constitutes one inequality for \( \tau_n \), which can be seen in List 1. The number of elements in \( S_i \) becomes huge especially when ratio \( p_i / p_a \) is large [4]. To avoid confusion, by scheduling point we mean every \( t \in S_i \), while the point which is tested in actual during analysis is denoted by actual-point, in rest of the paper. Clearly, an efficient technique would be the one which is capable of restricting the number of scheduling points as well as the number of actual points during online feasibility analysis.

**Hyper-Planes Exact Test:** To reduce scheduling points, E. Bini and G.C. Buttazzo provided a formulation, called Hyper-Planes Exact Test (HET) recently in [4] that reduces scheduling point for \( \tau_i \) from set \( S_i \) to a reduced set \( H_i(t) \). For any task \( \tau_i \), their test begins with \( p_i \) and expands its search space by

\[ H_i(t) = H_i \left( \left[ \frac{t}{p_i} \right] p_i \right) \cup H_i (t) \] (11)

Where \( H_i (t) = \{ t \} \). Two interesting observations can be made here:

**Observation 1:** To offer a tradeoff between complexity versus performance, the test can be tuned accordingly through a parameter \( \delta \) in the range of 0.1 - 1.0. Giving a lower value to \( \delta \) (up to 0.5), HET exhibits lower complexity (comparable to LL-bound), however it becomes sufficient condition only, in this case. Putting \( \delta = 1 \) makes HET both
necessary and sufficient at the expense of high complexity [4]. Though efficient when \( \delta \) has lower values, however, care is required to use HET in exact form \((\delta = 1)\). In such case, the cordiality of \( H_i(t) \) can reach 2 in worst case, thus, useless in exact form, when applied to a larger task set i.e., \( n \to \infty \).

**Observation 2:** A convincing argument for the superiority of any algorithm over exact test would be that \( \exists S_{\text{reduc.}} \subseteq S \). On one hand, HET efficiently provides a reduced set \( H_i(\rho) \subseteq S \), while on the other, this result has a potential drawback of testing the same point repeatedly due to its recursive nature i.e. because of calling \( \text{WorkLoad()} \) function recursively.

Since this paper is focused on exact tests, therefore, by HET we mean HET in its exact form \((\delta = 1)\). Below we re-write the same pseudo code given in [4] for reference.

```plaintext
procedure Boolean RMTTest ( \( r_n \))
    int i;
    for all \( i = 0; i < n; i + + \) do
        if \( c_i + \text{WorkLoad}(i - 1, \rho) \) then
            return false;
        else
            return true;
        end if
    end for
end procedure
```

/* Recursive function */

double last \( \Psi \) [BIG_ENOUGH];
double lastWorkLoad [BIG_ENOUGH];
function double WorkLoad(int i, double b) {
    int f,g;
    double branch0, branch1;
    if (i < 0) then
        return 0;
    end if
    if (p < last \( \Psi \) [i]) then
        return lastWorkLoad [i];
    end if
    f = \[ \begin{bmatrix} b \\ P_i \end{bmatrix} \];
    g = \[ \begin{bmatrix} b \\ P_i \end{bmatrix} \];
    branch0 = b - f(\( p_i - c_i \)) + WorkLoad(i - 1, f \times P_i - 1);
    branch1 = g \times c_i + WorkLoad(i - 1, b);
    last \( \Psi \) [i] = \delta;
    lastWorkLoad[i] = \min(branch0, branch1);
    return lastWorkLoad[i];
} end function
```

The above recursive function calling has implicit tendency for manipulating a large sequence of points (up to \( 2^n \) for \( r_n \)). To make the point clear, the feasibility of a task set given in Table 2 is answered in the light of TDA and HET respectively, where more actual-points are needed to be evaluated by HET than TDA. The advantage of TDA over HET is shown in Table 3.

**Illustrative Example:** The RM scheduling points constructed by both TDA and HET are given in Table 2, where task feasibility has to be tested. At first glance, the advantage of HET is clear, as \( H_i \subseteq S \) and \( \sum_{k=1}^{\infty} | H_i | \leq \sum_{k=1}^{\infty} | S_k | \). However, the important results are given in Table 3. Results can be compared on relaxed task sets, where case \( \frac{t}{r_n} = 1 \) may happen, such formulation does not contribute any point to the intended search space [4]. In order to provide more appropriate task set for HET, we deliberately use the task set in Table 1 to avoid the case when \( \frac{t}{r_n} = 1 \).

In our example this case never arises, as \( p_i \) is not integer multiple of \( p_0 \) or \( p_i \). The difference between the number of scheduling points constructed and the actual-points for respective tasks, can be observed in Table 2 and 3 respectively (more actual-points (multi-set) for HET in Table 3 than set \( H_i \) in Table 2). With TDA, for task \( r_n \), \( S_n \) consists of a large number of scheduling points, however only 3 actual-points tested during analysis, as \( W_j(t) \) is satisfied at \( t = 8 \) (third point in set \( S_k \)). In contrast, for the same task \( r_n \), though \( H_i \) consists of fewer points; it starts testing feasibility at multiple points at run-time (Table 3). We graphically show this behavior in Fig. 1.

From Eq.(11), we can see that for task \( r_n \), \( H_i \) is expanding in descending order, it commences feasibility test with largest \( t \) among all points in set \( H_i \).
The small circles in Fig. 1 reflects points which are repeated due to recursive function calling (WorkLoad), while filled circles shows \( t \in H_i \). As long as number of scheduling points and actual-points are concerned, for the referred \( \tau_i \), TDA offers 54\% improvement over HET. Eventually, for larger task set \((n \rightarrow \infty)\), we can conclude that TDA suppress HET in terms of time complexity as \( O(n^{\frac{1}{2}}) < O(2^n) \).

Similarly, when \( p_j = p_j \forall j \leq n \), \( n \) multiple points are contributed to search space produced by HET. Another interesting case would be the feasibility analysis of a low computation task set where
\[
\sum_{i=1}^{n} c_i \leq p_{\text{min}}.
\]
In this case, applying HET, results in a large array of scheduling points due to union operator involved, which is shown in Eq. (11). For such task sets, it is sufficient to test only \( n \) number of actual-points by TDA, as feasibility is positively answered at \( \tau_{i,1} \sum_{j=1}^{n} c_i \frac{p_j}{P_j} \leq p_j \)
holds true \((p_{\text{min}} \text{ being the smallest } t \in S_n \text{ is the first actual-point to be tested for any task } \tau_i)\).

**Enhanced Time Demand Analysis**: Using Theorem 1, it is equally important for a low priority task to be tested in interval \([0, p_j]\) despite the fact that there is a fair chance of not fulfilling the cumulative workload constituted by current task because interference from the higher priority tasks \((\tau_{i,1} \text{ to } \tau_{j})\) is also added to computation demands of current task \( \tau_i \). e.g. any false inequality for \( \tau_i \) through Eq.(9) also remains false with Eq.(10) for \( \tau_j \). We avoid this redundancy of points by proposing that when the time demand for task \( \tau_i \) is not fulfilled at any point \( t \), then it becomes unnecessary to test feasibility of the lower priority tasks \((\tau_{i,1} \text{ to } \tau_j)\) at the same point \( t \), as demand must remains unsatisfied at point \( t \).

**Mathematically**:
\[
\sum_{j=i}^{n} \left[ \frac{t}{P_j} \right] c_j + c_{\text{lower}} > t
\]
is always true, where \( c_{\text{lower}} \) is the execution demand of tasks whose priority is lower than \( \tau_i \), since \( c_{\text{lower}} > 0 \forall c_{\text{lower}} \in [c_{\tau_{i+1}}, c_j] \).

To reduce the intended search space, we present some definitions and theorems, which eventually lead us to the main contribution of this section.

**Theorem 2**: For any given periodic tasks set \( \tau_i \), scheduled by RM, task \( \tau_i \) has a set of RM scheduling points \( S_{\tau_i} \), \( S_{\tau_i} \subseteq S_{\tau_{i+1}} \).

**Proof**: To prove this theorem, we must prove that if there exist a scheduling point \( t \in S_{\tau_j} \), then \( t \in S_{\tau_{i+1}} \) also holds good. For arbitrary element of \( S_{\tau_j} \), it follows that \( t = a p_{\tau_j} \), where \( a \in \{1,..., \lfloor p_{\tau_j} / p_j \rfloor \}, j \in 1,..., i \) and \( b \in 1,..., i \). According to RM, tasks priorities are inversely proportional to their periods, tasks \( \tau_i \) has higher priority than \( \tau_{i+1} \), as \( p_{\tau_{i+1}} > p_i \), so it can be seen that \( a \in \{\lfloor 1,..., p_{\tau_{i+1}} / p_j \rfloor \} \) and \( j \in 1,..., i+1 \), hence \( t \in S_{\tau_{i+1}} \), and therefore \( S_{\tau_i} \subseteq S_{\tau_{i+1}} \).

**Definition 1: False Point (FP)**: A point \( t \in S_{\tau_i} \) is said to be false for any task \( \tau_i \) if it satisfies the inequality constraint: \( W(t) > t \).

**Theorem 3**: Given a set of \( n \) periodic tasks \( \{\tau_{i,1},...\} \) scheduled by RM, every false point for \( \tau_i \) must also be a false point for \( \tau_{i+1} \).

**Proof**: According to the definition of false point for task \( \tau_i; W(t) > t \), thus for task \( \tau_{i+1} \)
\[
W_{\tau_{i+1}}(t) = \sum_{j=1}^{\infty} \left[ \frac{t}{P_j} \right] c_j = \sum_{j=1}^{\infty} \left[ \frac{t}{P_j} \right] c_j + \sum_{j=1}^{\infty} \left[ \frac{t}{P_{j+1}} \right] c_j
\]
\[
= W(t) + c_{\tau_{i+1}} > W(t)
\]
As \( c_{\tau_{i+1}} > 0 \), therefore \( W_{\tau_{i+1}}(t) > t \) and hence \( t \) is also a false point for \( \tau_{i+1} \).

**Theorem 4**: Given a set of \( n \) periodic tasks \( \{\tau_{i,1},...\} \) scheduled by RM, every false point for \( \tau_i \) is also a false point for task having priority lower than \( \tau_i \).

**Proof**: This theorem can be directly deduced from Theorem 3.
Definition 2: Chained Point: A false point \( t \) for task \( \tau \) becomes chained point for the task with priority lower than \( \tau \). We represent all the false points contributed by \( \tau \) with a set \( X' \) such that \( X' \) becomes a set of chained points for \( \{ \tau_1, ..., \tau_n \} \).

Theorem 5: Given a set of \( n \) periodic tasks \( \{ \tau_1, ..., \tau_n \} \), \( \tau_i \) can be feasibly scheduled for all tasks phasings using RM if:

\[
L_i = \min_{t \in \mathcal{T}} \frac{W(t)}{t} \leq 1
\]

Where \( Z_i = S_i - X_{\tau_i} \).

By Extension \( X'_0 = \emptyset \)

Proof: The proof is straightforward, from Theorem 4, we known that, if \( t \) is a false point for \( \tau_i \), then it must also be a false point for \( \tau_j \).

Consequently to calculate \( L_i \), we confined our search to a reduced set \( Z_i \subseteq S_i \) by excluding all chained points.

Theorem 6: The entire task set \( \tau \) is schedulable for all task phasings using RM if:

\[
L = \max_{1 \leq i \leq n} L_i \leq 1
\]

We have replaced \( S_i \), given in Theorem 1 by \( Z_i \), which immediately reflects the reduced complexity of our technique \( Z_i \subseteq S_i \); allows us to search reduced number of inequalities for determining task feasibility. Task feasibility is drawn immediately when \( W_i(t) \leq t \) is satisfied at the smallest \( t \in Z_i \). We use the term Enhanced Time-Demand Analysis (ETDA) to refer our scheme hereafter in this paper. The effectiveness of ETDA becomes more prominent when it is applied to a large task set.

Pseudo Code for ETDA Can Be Written As Follows:

Procedure ETDA (\( \tau \))

\( X_i = \emptyset \)

For all \( \tau \in \tau \)

• Compute \( S_i = \{a \in \{0, 1, ..., \alpha \} | b = 1, ..., \beta \} \)

\( Z_i = S_i - X_{\tau_i} \)

For all \( t \in Z \)

If \( (L_i \leq 1) \) then

\( \tau \) is schedulable, break. else update \( X_i \) and end if end for if \( (L_i > 1) \) then \( \tau \) is feasible Else is infeasible; end if end for end procedure.

A graphical explanation for the identification of FP is given in Fig. 2. The symbol \( i \) shows the time period of the current task \( \tau \), while \( \tau_i \) denotes the testing points contributed from higher priority tasks. Let's assume task \( \tau_i \) is infeasible at point \( t = p_i \) (the deadline for the highest priority task \( \tau_i \)) or algebraically:

\[
\sum_{j=1}^{n} c_j \frac{\tau_i}{z_i} > t
\]

This fact makes it unnecessary to analyze feasibility of lower priority tasks (from \( \tau \) to \( \tau \)) at \( i = p \) execution demand of lower priority task is also added to the left side of already false inequality constraint i.e.

\[
\sum_{j=1}^{n} c_j \frac{\tau_i}{z_i} + c_i < t
\]

never holds true at \( i = p_i \), since the workload cannot be reduced while considering additional lower priority tasks.

We mark \( p_i \) as FP and avoid this inequality constraint to be tested by lower priority tasks. The same treatment is applied to all FP that arise as ETDA proceeds.

In Fig. 3, we visualize Theorem 5. The feasible interval \( [0, p_i] \) for any task \( \tau \) is divided into three sub portions. The filled rectangle represents all candidate scheduling points \( Z_i \). At each point \( t \in Z_i \), \( L_i \) is evaluated, which is either true or false. A true answer declares feasibility of \( \tau_i \), otherwise \( t \) is marked as FP. The thin horizontal rectangle with crossed lines depicts the false region, which consists of all FP, where task feasibility of current task \( \tau \) is definitely false. False region becomes larger and larger as the feasibility test proceeds, due to emergence of more FP. Once task feasibility is confirmed, our algorithm skips further search space, which is denoted by blank rectangle in Fig. 3.

Experimental Results and Analysis: In this section, we evaluate the performance of ETDA. In order to make a comparison, both TDA and HET are also implemented.
For our experiments, we generate random task periods in the range of [10,1000] with uniform distribution. Similarly, for corresponding task execution demands, random values are taken in the range of [1, p], also with uniform distribution. Tasks priorities are assigned according to RM assignment algorithm. The work is compared, in light of three performance evaluation criterions i.e. i) the number of RM scheduling points used, ii) the number of actual-points tested and iii) the execution time, which is the intended contribution of this paper.

Number of Sheduling Points: In this section, we demonstrate the effectiveness of ETDA in terms of reducing the number of scheduling points. Fig. 4 provides a comparison among the tests by counting the number of scheduling points i.e. $\sum_{i=1}^{n} S_i, \sum_{i=1}^{n} H_i, \sum_{i=1}^{n} Z_i$ for TDA, HET and ETDA respectively. Together, a series of periodic tasks is generated by varying the range from 5 to 50 with the increase of 5 tasks and average is taken after 200 iterations. We analyze all tests by varying system utilization form $ln(2)$ to 1.0. When tasks set size increases $(n \rightarrow \infty)$, more inequalities are needed to be tested as $p_i/p$ is becoming larger. Since $i$ is always feasible when $u_{opt} \leq ln(2)$, LH-bound would be sufficient to determine feasibility, however, we are intended to find feasibility through exact tests and hence the numbers of scheduling points are discussed. At higher system utilization, c, becomes larger and more points are needed to reach the conclusion (more inequalities to test). The graph of TDA and ETDA is increasing with increased utilization, since both commence testing i in ascending order. HET enjoys the advantage here of being non-increasing here, for any task $i$, it starts testing feasibility with largest element from H and therefore task feasibility is answered quickly. It can be seen that both HET and ETDA have effectively reduced the number of scheduling points as compared to TDA. This improvement is due to so-called reduced sets $(H_{i} \subseteq S_i$ and $(Z_i \subseteq S_i)$ required for any task $i$, by HET and TDA respectively.

Number of Actual Points: In this important experiment, we extract the number of actual-points, which are being tested by all necessary and sufficient tests discussed earlier in Section 3.1, 3.2 and 4 respectively. Results in Fig. 5 are obtained by performing computation over 200 task sets, each consist of 5 to 30 tasks. Here we also varied the total utilization from $n^{(2^{n-1})}$ to 1.0. We observed that the number of actual-points tested is directly influenced by variation in system utilization. The reason is that for higher utilization the execution demand of each task is high, since uniform distribution is applied. This fact makes the cumulative workload of individual task $i$, higher and its schedulability becomes possible at some later stage in the interval $[0, p_i]$. At lower system utilization $u_{opt} < ln(2)$, all tasks $i_1, i_2, ..., i_n$ are feasible, so feasibility of all n tasks is tested in sequence and thus, more actual-points are utilized. While for higher $u_{opt}$ there exist some unschedulable task $i_1 < i < n$, where feasibility of $i$ is determined early and therefore the feasibility tests quits immediately (skipping lower priority tasks). It is found that HET uses union operator and has implicit tendency for testing repeated points from intended search space obtained with Eq. (8).
Fig. 4: Required number of scheduling points

In contrast, on one hand, for feasible task set having \( u_{tot} \leq n(2^{m} - 1) \) both ETDA and TDA converges early (fewer inequalities required), while on the other, more actual-points are needed to be analyzed for higher utilization when \( u_{tot} \) (again in ascending order). Clearly, the improved performance of ETDA is due to \((Z \subset S)\), which is a direct conclusion of Theorem 5, that suppresses others in terms of testing reduced number of actual-points as shown in Fig. 4. For TDA and ETDA the number of scheduling points are always more than or equal to actual-points, since both are implemented non-recursively. In our experiment it is revealed that HET uses repeated points. This fact is witness by the difference in Fig. 4 and Fig. 5, where no agreement is followed between number of scheduling points and actual-points tested by HET. As mentioned before, though HET reduces the number of scheduling points in theory, it looses its effectiveness at run time by scanning redundant points from the search space due to its recursive implementation. In our proposed test, there is a direct mapping between Fig. 4 and Fig. 5.

**Execution Times:** The time a test takes to solve feasibility problem is an obvious criteria for measuring the performance of a given algorithm, especially when efficiency is the primary concern. This experiment supports our claim that only reducing number of scheduling points is useless unless actual number of points utilized is also handled carefully. The task set is constructed with the same criteria, as discussed earlier. The effect of changing system utilization is also observed. It is note that both TDA and ETDA increases with higher utilization, because more points are needed to be tested in this case, which eventually takes longer to
Fig. 5: Actual number of scheduling points tested

Fig. 6: Execution times of exact tests
conclude feasibility of the task set as given in Fig. 6. TDA’s graph is increasing according to utilization, since it start testing feasibility with smallest scheduling point in ascending order, that’s why at higher utilization the commutative workload can not be satisfied and more actual-points are needed to be evaluated. From Fig. 5 and 6, with increased number of actual-points, the taken by TDA also increases in Fig. 6. As expected ETDA takes negligible amount of time, because of skipping huge number false points. The non-increasing behavior of HET with increasing utilization is for certain, since number of actual-points are also reduced (Fig. 5).

In the light of Fig. 4, 5 and 6, it can be seen that, though the number scheduling point for HET is lowered (Fig. 4), the time taken by HET for the same task set is still high (Fig. 6). For HET Fig. 5 and 6 are in agreement and not Fig. 4 and 6. Thus, it is concluded from the experimental analysis that restricting actual-points is more important for online analysis of periodic task set than just reducing scheduling points.

CONCLUSION

We addressed the basic question of testing feasibility of periodic task under fixed priority assignment algorithms by eliminating redundant points in feasibility analysis. To reach the conclusion, merits and demerits of both SC and NSC are discussed. As a background work, we have shown the impracticality of exact tests for large tasks set to be performed online for real time systems. Similarly, a recently proposed efficient feasibility test is critically evaluated and its shortcomings are explored both theoretically and experimentally. Lastly, we have proposed ETDA, a state of the art solution for testing online feasibility of real time systems employing RM scheduling.

We evaluated the correctness and goodness of ETDA by means of mathematical proofs and simulation results. The experiment aimed at testing the algorithm under different performance conditions and comparing its results with previous solutions. The experimental results obtained, show the effectiveness of ETDA in providing an efficient online analysis for periodic tasks and validated our theoretical results.

ACKNOWLEDGMENTS

This work is jointly supported by COMSATS Institute of Information Technology under faculty development program and, the research collaboration between the Chinese Academy of Sciences and the Royal Society of the United Kingdom (Grant Number: 20030389, 20032006).

REFERENCES


