Revisiting Fixed Priority Techniques

Nasro Min-Allah[1,2,3], Wang Yong-Ji[3], Xing Jian-Sheng[1,3], and Junxiang Liu[3]

[1] Graduate University, Chinese Academy of Sciences, Beijing 100039, P.R. China
[2] Department of Computer Sciences, COMSATS University, 44000, Pakistan
[3] Institute of Software, Chinese Academy of Sciences, Beijing 100080, P.R. China
{nasar,ywang,jiansheng,liu}@itechs.iscas.ac.cn

Abstract. Discrete scheduling is preferred over continuous scheduling for preemptive scheduling problems, however, classical continuous schedulability tests can not work effectively with discrete scheduling. This area of integrating discrete scheduling with continuous schedulability tests remains unexplored. Two contributions are made in this paper; firstly, an empty-slot method is introduced and extended to discrete schedulability analysis and secondly, an efficient exact feasibility test is proposed that has lower complexity in contrast to current feasibility tests in terms of reducing the number of scheduling points, where task feasibility is analyzed.

Keywords: Real-Time Systems, Fixed priority Scheduling, Discrete Scheduling, Rate Monotonic Analysis, Time Demand Approach.

1 Introduction

Preemptive scheduling- a task may be preempted and resumed at some later stage- has become a mature field and a lot of literature is available [1], [2], [5]. Real-time systems implement preemptive scheduling through the priority scheme: higher priority job preempts a lower priority job. The system at each time instant assigns a priorities to active jobs and allocates the processor to the one acquiring the highest priority.

Currently, two approaches are available to implement preemptive scheduling algorithms in hard real-time systems: event-driven and timer-driven [6]. As long as preemption is concerned it can only occur at specified discrete time intervals for timer-driven approach, while for event-driven approach, preemptions occur when external interrupts arrive. In both case, interrupts are handled only, when current instruction is executed and system status is stored for later reference. As each instruction consumes a fixed number of CPU's clock time, a preemption only occurs at specified discrete time interval for event-driven approach. So it is a reasonable abstraction that preemptions occur only at discrete time intervals for hard real-time systems [7]. It is the preemption that makes scheduling either continuous or discrete. As integer values are easier to be implemented and operated than real numbers, discrete scheduling is preferred over continuous scheduling in hard real-time computer systems.

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 134–145, 2007.

Schedulability analysis is crucial for hard real-time systems to maintain system timing constraints. A lot of work has been done on Rate Monotonic (RM) and Deadline-Monotonic (DM) analysis [4], [9], [13], [16], [17]. However, most of the work done is focused on i) continuous schedulability analysis and ii)their complexity is pseudo-polynomial (in exact form). In this paper, we categorically address the above issues in Section 2 and 3, respectively.

In [10], Santos et al. proposed an empty-slot method and applied it to analyze the schedulability of communication tasks in real-time LANs. However, their task system is restricted to implicit-deadline and the execution time of each task must be equal to 1. Subsequently, Santos and Orozco [11] extended this method to MTSP (Multiple Tasks-Single Processor) systems and removed the restriction on the execution time of tasks but the task system is still restricted to implicit-deadline and the method is applicable to the RM schedulability analysis only.

Up to now, there has been no general method for analyzing discrete schedulability. We address this issue in Section 2. where we first analyze the empty-slot method and extend it for general discrete schedulability analysis. Then we analyze the discrete schedulability with a more general, constrained-deadline, synchronous task system and propose a new discrete schedulability test, which can be applied to all static-priority scheduling algorithms. This paper also presents a novel technique to handle the complexity of exact tests by reducing the number of inequalities which are needed to be tested otherwise for determining system feasibility, in Section 3. Finally, conclusions are drawn in Section 4.

2 Discrete Scheduling by Empty-Slot Method

2.1 Empty-Slot Method

Empty-slot method [10] is an appropriate approach for general discrete schedulability analysis. The main idea is: time is slotted and the duration of one slot is taken as unit of time. For any task τ_i, its parameters c_i, p_i and d_i are mutually commensurable and are assumed to be positive integers. Slot 1 is the first slot. Initially, all slots are empty. Each task acquires its slots according to the scheduling algorithm. Authors in [10] represented a node by a task, which must transmit its messages in one slot. The network is specified as a task set $\tau = (1, p_1), ..., (1, p_n)$. The expression $W_n(t) = \sum_{h=1}^{i-1} \lceil \frac{t}{p_1} \rceil$ gives this worst case of load in the interval $[0, t]$. Set τ is said to be non-saturated iff $W_n(M) < M$, where M denotes the least common multiple of all tasks. It has been proved that τ is RM schedulable iff for $i = 2, ..., n, \sum_{h=1}^{i-1} \lceil \frac{t}{p_h} \rceil < 1$ and $p_i \geq e_{1(i-1)} =$ least $t : t = 1 + \sum_{h=1}^{i-1} \lceil \frac{t}{p_n} \rceil$, where $e_{1(i-1)}$denotes the first empty slot of the higher priority $i-1$ tasks. There exist some constraints on the above task model i.e. tasks must have transmission time (execution time) equal to 1 and deadline equal to period.

Santos et al. extended this method to MTSP systems and the constraints on execution times were weakened by assuming any integer values [11]. However, the task model is still implicit-deadline synchronous. τ is RM schedulable iff $\forall i \in$

$2, ..., n, \sum_{h=1}^{i-1} \frac{c_h}{p_h} < 1$ and $p_i \geq e_{c(i-1)}$, where $e_{c(i-1)}$ denotes c_i-th empty slot of the higher priority $i-1$ tasks. From the above results, it is a reasonable conjecture that a similar result can be obtained for constraint-deadline synchronous task model when the timing constraint $\forall i, d_i = p_i$ is weakened with $\forall i, d_i \leq p_i$.

2.2 Discrete Static-Priority Schedulability Analysis

Preliminary Results. Let N denotes a natural number and $N^+ = N - \{0\}$. Let us define the function:

$$f(t) = \sum_{i=1}^{n} c_i \lceil \frac{t}{p_i} \rceil \tag{1}$$

Since $f(t)$ gives the maximum load in the interval $[1, t]$. Some conclusions can be drawn:
A. $f(t_2) - f(t_1)$ is the workload in the interval $[t_1, t_2]$
B. $f(t+1) - f(t)$ is the workload generated at the beginning of slot $t+1$. If the system is schedulable and $f(t+1) > f(t)$, then the interval $[t+1, t+(f(t+1)-f(t))]$ is full. If $f(t+1) = f(t)$ then slot $t+1$ can be empty.
C. if $f(t) \geq t$, slot t is full, otherwise slot t could be either full or empty.
It is obvious that $f(t)$ is non-decreasing monotonically. We can prove the following lamma :

Lamma 1

$$f(t_1 + t_2) = f(t_1) + f(t_2) - j, j \in \{0, 1, ..., \sum_{i=1}^{n} c_i\} \tag{2}$$

Proof. As

$$\lceil (a+b)/c \rceil = \begin{cases} \lceil a/c \rceil + \lceil b/c \rceil \\ \lceil a/c \rceil + \lceil b/c \rceil - 1 \end{cases} \quad \forall a, b, c \in N^+$$

$$\lceil \frac{(a+b)}{c} \rceil = \lceil \frac{a}{c} \rceil + \lceil \frac{b}{c} \rceil - j, j \in 0, 1$$

$$\text{then} f(t_1 + t_2) = \sum_{i=1}^{n} \lceil \frac{t_1 + t_2}{p_i} \rceil c_i$$

$$= \sum_{i=1}^{n} \{ \lceil \frac{t_1}{p_i} \rceil c_i + \lceil \frac{t_2}{p_i} \rceil c_i - j_i c_i \}$$

$$= \sum_{i=1}^{n} \lceil \frac{t_1}{p_i} \rceil c_i + \sum_{i=1}^{n} \lceil \frac{t_2}{p_i} \rceil c_i - \sum_{i=1}^{n} j_i c_i$$

$$= f(t_1) + f(t_2) - j$$

where $j = \sum_{i=1}^{n} j_i c_i$, as $j_i \in \{0, 1\}, j \in \{0, 1, ... \sum_{i=1}^{n} c_i\}$. We can see that τ repeats its behavior every M slots and to study its behavior it suffices to analyze it in the interval $[1, M] \leq M$. If τ is schedulable, the workload generated by τ

in the interval $[1, M]$ must execute to completion, therefore $f(M) \leq M$ is a necessary condition for τ to be schedulable. If $M - f(M) > 0$, then it gives the number of empty slots in the interval $[1, M]$ and the system is said to be non-saturated. If $M - f(M) = 0$, the system has no empty slots in the interval $[1, M]$ and is called saturated.

Theorem 1. *For a non-saturated system under any static-priority scheduling algorithm, the i-th empty slot e_i is the minimum value such that $t = i + f(t)$, $t \in [1, +\infty], \forall i \in N^+$.*

Proof. As e_i is the i-th slot, there are $i-1$ empty slots in the interval $[1, e_i - 1]$. Since e_i is empty, workload generated by all tasks in the interval $[1, e_i - 1]$ can execute to completion. Therefore, $f(e_i-1) = (e_i-1)-(i-1) = e_i-i$. No workload is generated at the instant e_i, $f(e_i) = (e_i - 1)$, namely $e_i = i + f(e_i)$. Here we prove that e_i is the minimum $t|t = i + f(t)$. Since $\forall t < e_i$, there are at most $i-1$ empty slots in the interval $[1, t]$, then $f(t) \geq t - (i - 1) > t - 1 \Longrightarrow t < i + f(t)$. Therefore, e_i is the minimum $t|t = i + f(t)$.

Theorem 2. *For a non-saturated system under any static-priority scheduling algorithm, there is*

$$e_{i+1} - e_i \leq e_a, \forall i \in N, a \in N^+ \tag{3}$$

Proof. To prove Theorem 2 it suffices to prove $\forall i, e_{i+a} \in [e_i + 1, e_i + e_a]$. Since e_i is empty, the workload generated in the interval $[1, e_i - 1]$ has been dealt with. According to Lemma 1 and conclusion A, the workload is:
$W = f(e_i + e_a) - f(e)i) = f(e_i) + f(e_a) - j - f(e_i) = f(e_a) - j$
From Theorem 1, there is $f(e_a) = e_a - a$, so $W = e_a - a - j$. The difference between the workload and the length of the interval is no less than a, so there are at least a empty slots in the interval. Therefore, $e_{i+a} \in [e_i + 1, e_i + e_a]$.

2.3 Discrete Static-Priority Schedulability Test

Theorem 3. *If task set τ_{n-1} is static-priority schedulable, when added a new low priority task τ_n, then τ is static-priority schedulable if and only if τ_{n-1} is non-saturated and $D_n \geq e_{c_n(n-a)}$.*

Proof. First we prove the sufficient condition: Task τ_n generates workload at instant $1, p_n + 1, 2p_n + 1$, and so on. To check the schedulability of τ it suffices to check if there are enough empty slots left to execute task τ_n in the interval $[kp_n + 1, kp_n + d_n]$. From Theorem 2, we know that $e_{c_n(n-1)} \geq e_{i+c_n(n-1)} - e_{i(n-1)}$. If there are empty slots in the first interval $[1, d_n]$ to execute task τ_n to completion, there must be enough empty slots in the following intervals $[p_n + 1, p_n + d_n], [2p_n + 1, 2p_n + d_n]$, and so on. As $d_n \geq e_{c_n(n-1)}$, there are enough empty slots in the interval $[1, d_n]$, to execute task τ_n to completion, so τ is schedulable. Next we prove the necessary condition. If τ is static-priority schedulable, then τ_{n-1} obviously is non-saturated and schedulable. As task τ_n's priority is the lowest, there must be $d_n \geq e_{c_n(n-1)}$. From the theorem above, we can easily get the following theorem.

Theorem 4. *A task set τ is static-priority schedulable if and only if for $i = 2, ..., n$,*

$$\sum_{j=1}^{i-1} \lceil \frac{c_i}{p_i} \rceil \leq 1 \text{ and } d_i \geq e_{c_i(i-1)} \tag{4}$$

Proof. This schedulability test includes two parts. The first one guarantees that τ_{i-1} is non-saturated and consequently it can incorporate another task, τ_i. The second part guarantees that there are sufficient slots to execute task τ_i to completion before its deadline. We refer to this test as empty-slot static-priority (ESSP) schedulability test in rest of the paper. For each task, we calculate and check if it's the c_i-th empty slot of the higher priority $i-1$ tasks. The maximum number of times to calculate free slot for each task is d_n . So the time complexity of ESSP is $O(n \times d_n)$. As d_n increases with n in the worst case, the time complexity depends not only on the dimension n of the problem but also the magnitude d_n of the data involved, so the time complexity of ESSP is pseudo-polynomial.

2.4 Comparisons with Classical Continuous Schedulability Tests

Audsley et al. [9] proposed a necessary and sufficient schedulability test for RM scheduling:

$$\forall i, 1 \leq i \leq n, WR_i \leq d_i \tag{5}$$

WR_i is the worst-case response time of τ_i and is given by the smallest $x \in N^+$ that satisfies the following recursive equation.

$$WR_i^0 = \sum_{j=1}^{i} c_j \tag{6}$$

$$WR_i^{l+1} = c_i + \sum_{j=1}^{i-1} \lceil \frac{WR_i^l}{p_j} \rceil c_j \tag{7}$$

The above procedure stops when the same value is found for two successive iterations of l, or when the deadline d_i is exceeded. Its time complexity is pseudo-polynomial[12]. For simplicity, we only compare ESSP with the technique presented in [9]. ESSP exhibit similar behavior as classical test, though they are obtained with different methods. Compared with the classical test, ESSP has many advantages. First, integer number is easy to be operated with than real number. Secondly, ESSP can easily solve many practical problems: Given τ, which is non-saturated and static priority schedulable, determine

- Whether $S(n+1)$ is also schedulable? (by adding a new low priority task)
- Up to what extent the execution time of a task in τ can be expanded while keeping the system feasible?
- What is the value of the execution time of the task that saturates the system?

Although the time complexity of ESSP is pseudo-polynomial i.e. $O(n \times d_n)$, normally d_n may not necessarily increase with n and the time performance of ESSP may not necessarily be so. Further more, we improve it to handle on-line admission control by the following methods:

(1) If τ_{n-1} is non-saturated, then $S(i)|i = 1, ..., n-2$ is also non-saturated.
(2) For $d_i \geq e_{c_i(i-1)}$ to hold, $e_{c_i(i-1)} = mint|t = c + i + \sum_{j=1}^{i-1} c_j \lceil \frac{t}{p_j} \rceil \geq \sum_{j=1}^{i} c_j$, we can check this inequality from the slot $\sum_{j=1}^{i} c_j$ instead of 1.
(3) If $c_i + \sum_{j=1}^{i-1} c_j \lceil \frac{t}{p_j} \rceil = t^* > t$, then the next slot to be checked is t^*.
(4) For each task, the number of ceiling calculations is not d_n, but increases from $\sum_{j=1}^{i} c_j$ to d_n.

Thus, the number of effective operations needed for ESSP is much less than $n \times d_n$. From the above analysis, we can see that ESSP is more convenient to be used and more efficient to tackle practical problems than classical continuous schedulability tests.

3 Enhanced Deasibility Analysis

3.1 Previous Results on Exact Test

Time Demand Approach. For validating feasibility problem, both necessary and sufficient condition (NSC) based tests are proposed in literature [4], [8], [9], [13], [14], [17]. Recently, authors in [3], [4] extended the work illustrated in [8] by proposing an exact feasibility test that reduces the number of scheduling points. Our results show that reducing the number of scheduling point does not necessarily means lowering the number of inequalities in actual.

To determine whether a task can meet all its deadlines, we compute the total demand for processor time by a task τ_i at time t, as

$$W_i(t) = c_i + \sum_{j=1}^{i-1} \lceil \frac{t}{p_j} \rceil c_j \tag{8}$$

A periodic task τ_i is feasible if we find some $t \in [0, d_i]$ satisfying

$$L_i = \min_{0<t\leq d_i} (W_i(t) \leq t) \tag{9}$$

As t is a continuous variable, there are infinite numbers of points to be tested. The entire task set τ is feasible iff

$$L = \max_{1\leq i\leq n} \{ \min_{0<t\leq d_i} \frac{W_i(t)}{t} \} \leq 1 \tag{10}$$

The first attempt to limit the infinite number of points in interval $t \in [0, t]$ is made by authors in [8]. The authors in [8] show that $W_i(t)$ is constant, except at

finite number of points when tasks are released, called rate monotonic scheduling points. Consequently, to determine whether τ_i is schedulable, we need to compute $W_i(t)$, only at multiples of $\tau_i \leq \tau_j, 1 \leq j \leq i$. specifically, let

$$S_i = \{ap_b | b = 1, ..., i; a = 1, ..., \lfloor d_i/p_b \rfloor\} \tag{11}$$

We conclude that an dividual task is feasible iff the following equation is true.

$$L_i = \min_{t \in S_i} \frac{W_i(t)}{t} \leq 1 \tag{12}$$

L_i is needed to be analyzed only at a finite number of points i.e. S_i. In rest of the paper, we represent this technique by TDA (Time Demand Approach). For any task τ_i , the number of elements in set S_i is of particular interest. Every element means testing an inequality constraint for finding schedulability of τ_i at run time. The number of elements in S_i becomes huge especially when ratio p_n/p_1 is large [3]. Clearly, an efficient technique would be the one which is capable of reducing the number of inequalities testing during online feasibility analysis.

Hyper-Planes Exact Test. To reduce scheduling points, E. Bini and G. C. Buttaazo provided a formulation, called Hyper-planes Exact Test (HET) recently in [3], [4] that reduces scheduling point for τ_i from set S_i to a reduced set $H_i(t)$. For any task τ_i, their test begins with p_i and expands its search space by

$$H_i(t) = H_{i-1}(\lfloor \frac{t}{p_i} \rfloor d_i) \cup H_{i-1}(t) \tag{13}$$

where H_0(t)={t}.

3.2 Deadline Monotonic Analysis Improved

The necessary and sufficient condition (NSC) for the schedulability of τ_1 is that $c_1 \leq p_1$. Using Equation 8, it is equally important for a low priority task to be tested in interval $[0, p_1]$ 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 (τ_{i+1} to τ_n) is also added to computation demands of current task τ_i. We avoid this redundancy of points by proposing that when the time demand for task τ_i is not fulfilled at some point t, then

$$\sum_{j=1}^{i} \lceil \frac{t}{d_j} \rceil c_j + c_{lower} > t \tag{14}$$

is always true, where c_{lower} is the execution demand of tasks whose priority is lower than τ_i, since $c_{lower} > 0 \ \forall c_{lower} \in [c_{i+1}, ...c_n]$.

To reduce the intended search space, we present some definitions and theorems, which eventually lead us to the main contribution of this section (Theorem 8).

Theorem 5. *-For any given periodic tasks set τ, scheduled by DM, task τ_i has a set of DM scheduling points $S_i : S_i \subseteq S_{i+1}$*

Proof. To prove this theorem, we must prove that if there exists a scheduling point $t \in S_i$, then $t \in S_{i+1}$ also holds good. For an arbitrary element of S_i, it follows that $t = ap_b$, where $a \in \{1, ..., \lfloor d_i/p_j \rfloor\}$, $j \in 1, ..., i$ and $b \in 1, ..., i$. According to DM, tasks priorities are inversely proportional to their deadlines, tasks τ_i has higher priority than τ_{i+1} as $d_{i+1} \geq d_i$, so it can be seen that $a \in \{1, ..., \lfloor d_{i+1}/p_j \rfloor\}$, $j \in 1, ..., i+1$ and $b \in 1, ..., i+1$, hence $t \in S_i$ and therefore $S_i \in S_{i+1}$ holds true.

Definition 1 (False Point). *DM scheduling point t, is said to be false point for any task τ_i if it satisfies the inequality constraint:$W_i(t) > t$.*

Theorem 6. *Given a set of n periodic tasks $\{\tau_1, ..., \tau_n\}$ scheduled by DM, every false point for τ_i must also be a false point for τ_{i+1}.*

Proof. According to the definition of false point for task τ_i ;$W_i(t) > t$, thus for task τ_{i+1}

$$W_{i+1}(t) = \sum_{j=1}^{i+1} \lceil \frac{t}{d_i} \rceil c_j = \sum_{j=1}^{i} \lceil \frac{t}{d_i} \rceil c_j + \lceil \frac{t}{d_{i+1}} \rceil c_{i+1}$$
$$= W_i(t) + c_{i+1} > W_i(t) \tag{15}$$

As $c_{i+1} > 0$, therefore $W_{i+1}(t) > t$ and hence t is also a false point for τ_{i+1}.

Theorem 7. *Given a set of n periodic tasks $\{\tau_1, ..., \tau_n\}$ scheduled by DM, every false point for τ_i is also a false point for task having priority lower than τ_i.*

Proof. This theorem can be directly deduced from Theorem 6.

Theorem 8. *Given a set of n periodic tasks $\{\tau_1, ..., \tau_n\}$, τ_i can be feasibly scheduled for all tasks phasings using DM iff*

$$L_i = \min_{t \in Z_i} \frac{W_i(t)}{t} \leq 1 \tag{16}$$

where $Z_i = S_i - X_{i-1}$. By extension $X_0 = \emptyset$.

Proof. From Theorem 7, we known that, if t is a false point for τ_{i-1} then it must also be a false point for τ_i. Consequently to calculate L_i, we confine our search to a reduced set $Z_i \subseteq S_i$, by excluding all chained points.

The whole set τ is feasible iff

$$L = \max_{1 \leq i \leq n} L_i \leq 1 \tag{17}$$

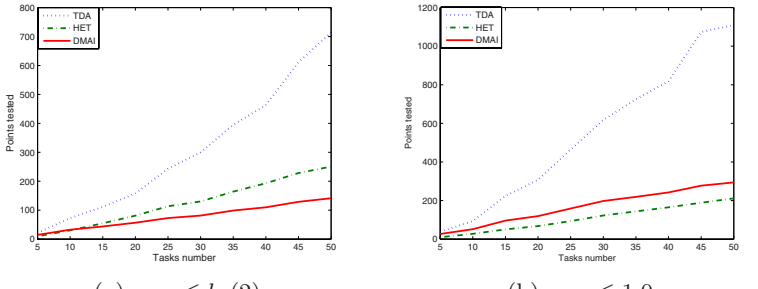
We replace S_i by Z_i, which immediately reflects the reduced complexity of our technique $Z_i \subseteq S_i$; allows us to search reduced number of inequalities for determining task feasibility. We use the term Deadline Monotonic Analysis Improved (DMAI) to refer our scheme hereafter in this paper. The effectiveness of DMAI becomes more prominent when it is applied to a large task set.

3.3 Experimental Results

In this section, we evaluate the performance of DMAI. In order to make a comparison with DMAI, 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 $[1, p_i]$, also with uniform distribution. Together, a series of periodic tasks is generated by varying the range from of 5 to 30 with increase of 5 tasks and average is taken after 200 iterations. Results are obtained under varied system utilization, however, due to space limitations, only two are shown for each experiment in the following. The work is compared, in light of two performance evaluation criteria i.e. the number of DM scheduling points used and, the number of inequalities tested , which is the intended contribution of this paper.

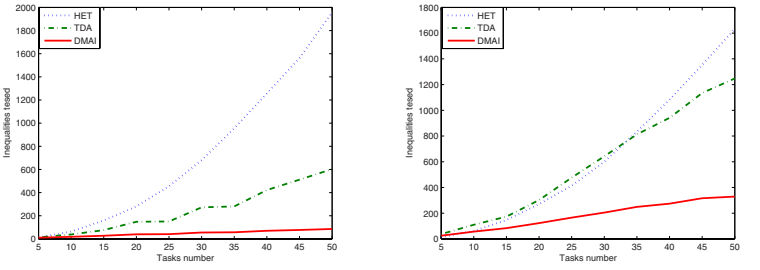
Number of Scheduling Points. In this section, we demonstrate the effectiveness of DMAI in terms of reducing the number of scheduling points. Fig. 1 provides a comparison among the tests by counting the number of points that are actually utilized before feasibility is concluded. We analyzed all tests by varying system utilization from $ln(2)$ to 1.0. When tasks set size increases i.e. $n \rightarrow \infty$ more inequalities are needed to be tested as p_n/p_1 is becoming larger. It is found that HET uses union operator and has implicit tendency for testing repeated values from intended search space obtained with Equation 13, we plot only unique values here. It can be seen that both HET and DMAI have effectively reduced the number of scheduling points as compared to TDA. This improvement is due to generally reduced sets $(H_{i-1}(p_i) \subseteq S_i)$ and $(Z_i \subseteq S_i)$ required for any task τ_i by HET and TDA respectively.

Number of Inequalities. In this important experiment, we extract the number of inequalities, which are being tested by all necessary and sufficient tests discussed earlier in Section 3.1 and 3.2 respectively. We observe that the number of inequalities tested is directly influenced by variation in system utilization. As



(a) $u_{tot} \leq ln(2)$ (b) $u_{tot} \leq 1.0$

Fig. 1. Effect of utilization on average number of scheduling points



(a) $u_{tot} \leq ln(2)$ (b) $u_{tot} \leq 1.0$

Fig. 2. Advantage of straight forward approaches over recursive technique

shown in Fig. 2, the little variation in behavior of HET under varied system utilization is mainly due of its dependency on task periods. In contrast, both DMAI and TDA equally exploit the execution demands of individual tasks and, are more inclined towards total system utilization. On one hand, for feasible task set having $u_{tot} \leq \ln(2)$ both DMAI and TDA converges early, while on the other, more points are needed to be analyzed for higher utilization when $u_{tot} \leq 1$. Clearly, the improved performance of DMAI is due to $(Z_i \subseteq S_i)$ which is a direct conclusion of Theorem 8 and, suppresses others in terms of testing reduced number of inequality constraint as shown in Fig. 2. It can be seen that, both TDA and HET use repeated points. This fact is witnessed by the difference in Fig. 1 and Fig. 2; there is a mismatch between number of scheduling points and inequalities tested for TDA and HET. As mentioned earlier, 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 approach, there is a direct mapping between Fig. 1 and Fig. 2.

4 Conclusions and Future Work

We first introduced a schedulability analysis method using empty-slot for discrete scheduling. The method is then applied to static-priority scheduling and a new efficient discrete schedulability test is proposed. The analysis confirms that, empty-slot method is an efficient discrete schedulability analysis method and, can be generalized to allow more general task model, in particular when tasks may have deadlines larger than periods. The analysis can also be applied to the case in which task synchronization must be considered or when the priority ceiling protocol is used.

In addition to above, we have proposed DMAI, a state of the art solution for testing online feasibility of real time systems employing DM scheduling, which has much lower complexity than current feasibility tests. We evaluated the correctness and goodness of DMAI 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 DMAI 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, the National Natural Science Foundation of China (Grant Number: 60373053) and the research collaboration between the Chinese Academy of Sciences and the Royal Society of the United Kingdom (Grant Number: 20030389, 20032006).

References

1. Liu, J.W.S.: Real Time Systems. Prentice Hall, Englewood Cliffs (2000)
2. Krishna, C.M., Shin, K.G.: Real-Time Systems. McGrawHill, New York (1997)
3. Bini, E., Buttazzo, G.C.: The Space of Rate Monotonic Schedulability. In: Proceedings of the 23th IEEE Real-Time Systems Symposium, pp. 169–177 (2002)
4. Bini, E., Buttazzo, G.C.: Schedulability Analysis of Periodic Fixed Priority Systems. IEEE Transactions on Computers 53(11), 1462–1473 (2004)
5. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard real-time environment. J. of the ACM 20(1), 40–61 (1973)
6. Katcher, D.I., Arakawa, H., Strosnider, J.K.: Engineering and analysis of fixed priority schedulers. IEEE Trans. On Software Engineering 19(9), 920–934 (1993)
7. Baruah, S., Mok, A., Rosier, L.: Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. Real-Time Systems 2, 301–324 (1990)
8. Lehoczky, J.P., Sha, L., Ding, Y.: The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior. In: Proceedings of the IEEE Real-Time System Symposium, pp. 166–171 (1989)
9. Audsley, N.C.: Deadline monotonic scheduling, Report YCS.146, Depart. of Comput.Sci., University of York (1990)
10. Santos, J., Gastaminza, M.L., Orozco, J., Picardi, D., Alimenti, O.: Priorities and protocols in hard real-time LANs. Computer and Commun. 14(9), 507–514 (1991)
11. Santos, J., Orozco, J.: Rate monotonic scheduling in hard real-time systems. Information Processing Letters 48, 39–45 (1993)
12. Audsley, N.C., Burns, A., Richardson, M.F., Wellings, A.J.: Hard real-time scheduling: the deadline monotonic approach. In: Proceedings of 8th IEEE Workshop on Real-Time Operating Systems and Software, pp. 133–137 (1991)
13. Sjodin, M., Hansson, H.: Improved response-time analysis calculations. In: Proceedings of the 19th IEEE Real-Time Systems Symposium, pp. 399–409 (1998)
14. Joseph, M., Pandya, P.: Finding response times in a real-time system. The Computer Journal 29(5), 390–395 (1986)

15. Leung, J.Y.T., Whitehead, J.: On the Complexity of Fixed-Priority Scheduling of Periodic. Real-Time Tasks Performance Evaluation 2, 237–250 (1982)
16. Kuo, T.-W., Mok, A.K.: Load Adjustment in Adaptive Real-Time Systems. In: Proceedings of the IEEE Real-Time Systems Symposium, pp. 160–171 (1991)
17. Manabe, Y., Aoyagi, S.: A feasibility decision algorithm for rate monotonic and deadline monotonic scheduling. Real-Time Systems 14(2), 171–181 (1998)
18. Tindell, K.W., Bums, A., Wellings, A.J.: An extendible approach for analyzing fixed priority hard real-time tasks. Real-Time Systems Journal 6, 133–151 (1994)