International Journal of Electrical and Computer Engineering (IJECE)
Vol. 8, No. 1, February 2018, pp. 429 – 440
ISSN: 2088-8708, DOI: 10.11591/ijece.v8i1.pp429-440

Schedulability of Rate Monotonic Algorithm using Improved Time Demand Analysis for Multiprocessor Environment

Leena Das[1], Sourav Mohapatra[2], and Durga Prasad Mohapatra[3]
[1]Departement of Computer Science and Engineering, KIIT University, Bhubaneswar, Odisha, India
[2,3]Departement of Computer Science and Engineering, National Institute of Technology Rourkela, Odisha, India

Article Info

Article history:
Received: Sep 2, 2017
Revised: Dec 26, 2017
Accepted: Jan 10, 2018

Keyword:
RMA
Real Time System
Time Demand Analysis
Scheduling
Multiprocessor

ABSTRACT

Real-Time Monotonic algorithm (RMA) is a widely used static priority scheduling algorithm. For application of RMA at various systems, it is essential to determine the system's feasibility first. The various existing algorithms perform the analysis by reducing the scheduling points in a given task set. In this paper we propose a schedubility test algorithm, which reduces the number of tasks to be analyzed instead of reducing the scheduling points of a given task. This significantly reduces the number of iterations taken to compute feasibility. This algorithm can be used along with the existing algorithms to effectively reduce the high complexities encountered in processing large task sets. We also extend our algorithm to multiprocessor environment and compare number of iterations with different number of processors. This paper then compares the proposed algorithm with existing algorithm. The expected results show that the proposed algorithm performs better than the existing algorithms.

Corresponding Author:
Leena Das
Departement of Computer Science and Engineering
KIIT University, Bhubaneswar, Odisha
ldasfcs@kiit.ac.in

1. INTRODUCTION

Real-time systems are different from other types of systems in the sense that they must provide accurate results in both temporal and logical aspects. Apart from being able to carry out the required task, the system must do it within a certain period of time. For example, when a temperature sensor in a thermal plant gives a warning, then the system must turn on cooling mechanisms withing a certain interval of time barring which the plant's operation may fail. Here the response of the system must be correct as well as be completed in a given interval of time. A real-time system can be divided broadly into three spheres; the environment, the controller and the controlled object. The controller gets input from the environment and then provides information to the controlled object. The time it takes to provide the instruction is known as the *execution time*. The time after which a event repeats itself in the environment is known as the *period* of that event. Every event needs a certain time interval before which it has to be executed. This is known as the *deadline*.

Thus a real time-system's primary goal is to provide a scheduling algorithms where all the deadlines are met, taking into account the period and execution time. To accomplish this there are a number of algorithms primarily categorized into two categories: static priority and dynamic priority algorithms. Static priority algorithms are those which the priority assigned are static in nature. While in dynamic, the priorities changes dynamically. This paper concentrates on static priority scheduling, most specifically RMA.

Rate Monotonic Algorithm (RMA) is one of the most widely used and effective scheduling algorithm. RMA uses mathematical model of static priority based scheduling where the priority is the *period* of the tasks. It supports the intuition that the tasks that occur more frequently should be given higher priority. RMA was first proposed in [1]. Working of RMA depends on the periods of the tasks. For a given task set to be termed as feasible, it must be possible to be scheduled with a given algorithm, in this case, RMA. This feasibility tests are generally known as schedulability bounds. For an algorithm to work, it must be within certain limits. Schedulability bound provides this limit. The

work in [1] gave an initial feasibility bound. It was improved by Seto and Lehoczy in [3, 2] by the use of a time demand analysis function to make it an exact feasibility test. Our work focuses on this aspect of RMA scheduling. Various types of other bounds were also proposed. The concept of harmonic period was explored by Kuo et. al in [4] that showed the scheduablility of tasks satisfying the harmonic conditions. Our paper now analyses the initial method proposed in [3, 2], often termed as time demand analysis and implements it. We then propose an algorithm to improve this implementation.

The rest of the paper is organized as follows: Section 2 describes the background and general theory referenced in this paper. It also describes the existing feasibility bounds. Section 3 describes the time demand analysis for the uniprocessor systems with RMA scheduling and provides motivation of how it can be improved. Section 4 describes the proposed algorithm for schedulability analysis and also extends it to multiprocessors real-time systems. We discuss the implementation of the proposed algorithm and provide results. In Section 5, we analyze the different results and comparison with related work are done. Conclusion and future work are then shown in Section 6.

2. BASIC CONCEPT

Real-time systems can be divided into three broad categories: (1) *Hard real-time Systems*, (2) *Firm real-time Systems* and (3) *Soft real-time Systems*.

If the completion of task is not achieved within the deadline in a *Hard real-time* system then a system failure occurs. While if the system still can function with some degradation in performance if the deadlines are missed then the system is termed as *Soft real-time* System. Firm real-time systems are in between Hard and soft real-time systems. In this paper we have considered *hard real-time* systems i.e. all the deadlines must be met.

The event that determines a course of action is known as a task. On the occurrence of a task, the system does processing and responds accordingly. There are three types of tasks:

1. Periodic Tasks: These are the tasks that occur after after a specified interval of time. For example a sensor sending temperature data every 10 seconds. We say that this task is periodic with period of 10 secs.
2. Aperiodic Tasks: These are the tasks which can occur after any amount of time after the occurrence of the last instance, except immediately.
3. Sporadic Tasks: These are aperiodic tasks where the repetition period can be zero.

In our paper we deal with periodic tasks only. Now we describe the various notations used in our paper.

- C_i : Execution time of i_{th} task
- T_i : Period of i_{th} task
- U_i : Utilization of i_{th} task
- w_i : Time demand function value
- t : Current time point

Any other notations have been described as and when used.

2.1. Related Works

In this section we describe the schedubility analysis of RMA and the related work done. We describe the various schedulability bounds present. We introduce the Time Demand Analysis [2, 5], which was further improved in [6, 7]. We also survey some works that gave us direction to proceed in the work.

In this paper [6] the authors proposed a novel schedulability analysis for verifying the feasibility of large periodic task sets under the rate monotonic algorithm, when the exact test cannot be applied on line due to prohibitively long execution times. The proposed test has the same complexity as the original Liu and Layland bound but it is less pessimistic, so allowing to accept task sets that would be rejected using the original approach. The performance of the proposed approach is evaluated with respect to the classical Liu and Layland method, and theoretical bounds are derived as a function O(n) (the number of tasks) and for the limit case of n tending to infinity. The analysis is also extended to include aperiodic servers and blocking times due to concurrency control protocols. Extensive simulations on synthetic tasks sets are presented to compare the effectiveness of the proposed test with respect to the Liu and Layland method and the exact response time analysis.

This paper gave a bound that improved on the existing bound [1]. It was named as hyperbolic bound. This bound was found to have a linear complexity. This can be used as an extension prior to the calculation of time demand analysis.

In the paper [10] feasibility analysis of fixed priority systems has been widely studied in the real-time literature and several acceptance tests have been proposed to guarantee a set of periodic tasks. They can be divided into two main classes: polynomial time tests and exact tests. Polynomial time tests can efficiently be used for an on-line guarantee of real-time applications, where tasks are activated at runtime. These tests introduce a negligible overhead, when executed upon a new task arrival, however, provide only a sufficient schedulability condition, which may cause a poor processor utilization. On the other hand, exact tests, which are based on response time analysis, provide a necessary and sufficient schedulability condition but are too complex to be executed online for large task sets. As a consequence, for large task sets, they are often executed offline. In this paper, the authors propose a novel approach for analyzing the schedulability of periodic task sets on a single processor under an arbitrary fixed priority assignment. Using this approach, they derive a new schedulability test which can be tuned through a parameter to balance complexity versus acceptance ratio, so that it can be used online to better exploit the processor, based on the available computational power. The test is shown to be significantly faster than the current response time analysis methods. Moreover, the proposed approach offers an explanation of some known phenomena of fixed priority scheduling and could be helpful for further work on schedulability analysis.

The paper [9] is an extension of the previous paper [10]. It explores in detail and extends the work done by generalizing it to fixed priority systems rather than just rate monotonic algorithm. The name of the approach is HET. There were other properties explored but not related to RMA schedulability. Rest of the details and proofs are extensions.

In the paper [12], the authors discuss how the high computational complexity required for performing an exact schedulability analysis of fixed priority systems has led the research community to investigate new feasibility tests which are less complex than exact tests, but still provide a reasonable performance in terms of acceptance ratio. The performance of a test is typically evaluated by generating a huge number of synthetic task sets and then computing the fraction of those that pass the test with respect to the total number of feasible ones. The resulting ratio, however, depends on the metrics used for evaluating the performance and of the method for generating random task parameters. In particular, an important factor that affects the overall result of the simulation is the probability density function of the random variables used to generate the task set parameters. In this paper, the authors discuss and compare three different metrics that can be used for evaluating the performance of schedulability tests. Then, they investigate how the random generation procedure can bias the simulation results of some specific scheduling algorithm. An efficient method for generating task sets with uniform distribution in a given space was given and shown how some intuitive solutions typically used for task set generation can bias the simulation results.

This is the primary paper from which the generation of task set has been done. The task set must be generated as the capture of real-time datasets is very difficult. Generation of a uniform dataset is the primary objective of this paper.

RMA was first proposed by [1] in 1973 as an optimal scheduling algorithm for static priority task set. The priority was assigned to the periods of the tasks. For a task set $T(T_1, T_2, ..., T_n)$ period(T_i) $<$ period(T_j) $\implies$ priority(T_i) $>$ priority(T_j)

For validating the feasibility of a task set by determining whether it is schedulable or not, a variety of tests have been developed. The first universal feasibility bound for all types of scheduling systems is given by

$$\sum_{1}^{n} U \leq 1 \tag{1}$$

The sum of utilization of all the tasks in the task-set should be less than equal to 1. This gives the necessary upper bound for any scheduling algorithm including RMA. But this is not sufficient. We refer to this bound as *schedulability bound 1*. A tighter feasibility test was proposed in [1] which stated that a periodic static priority system is feasible if

$$\sum_{1}^{n} U \leq n(2^{1/n} - 1) \tag{2}$$

Where n is the total number of tasks in the task-set. The value tends to $ln(2)$ as n tends to ∞. This shows that in any sufficiently large task-set, if the total utilization is less than 0.693, then it can be scheduled. This, however, is a sufficient condition only. Even if the total utilization remains greater than this bound, it can still be static feasible. To take into account this factor, various necessary and sufficient tests were proposed [2, 5, 6, 7, 8, 16, 17]. The initial test proposed in [2, 5] was further improved in [6, 8]. These all tests remained pseudo polynomial. The proposed tests

can be divided into two types; iterative [5, 16] and as-per-task basis [2, 9, 10, 8]. The later analyses feasibility on task arrival times known as scheduling points. The former uses an iterative technique. Recently the work by Allah et. al. in [7] improved the work by Bini and Buttazzo in [6] by restricting the actual number of scheduling points. We now discuss the exact feasibility test upon which we propose our improvement.

2.2. Exact Schecduability Test

Phase I of a task is defined as the time when the first instance of the task is released into the system. Two tasks T_i and T_j are said to be of same phasing iff $I_i = I_j$. In the work [1] it was shown that in the scenario where the phasing of all the tasks are equal, it results in the longest response time. This is generally known as the *critical instant*. This scenario creates the worst case task set phasing and thus can be used as a criterion for the schedulability of the given task set. The idea can be re-framed as "*A periodic task set can be scheduled by a fixed priority scheduling algorithm if the deadline of the first job of each task is met while using that algorithm from a critical instant*". Since RMA is a fixed priority scheduling algorithm, the condition satisfies for it.

Let T be a task set $T(T_1, T_2, ..., T_n)$ with tasks having increasing periods (thus decreasing priority). As per RMA's priority property a task having lower period has to be scheduled before task of higher period. Thus a task T_i can only be affected by the tasks T_j where period(T_j) $>$ period(T_i). This gives the intuition that that while checking for the schedulability of the task only those task should be considered whose periods more than the current (or priority less). This ordering can be achieved by sorting the task set in ascending order of their period. When the task set is now processed, the tasks are encountered with decreasing priority. As the At the time of critical instance, a value is calculated. This value gives the estimate as to how much the system is feasible. These ideas were stated mathematically in [2] as follows

$$w_i(t) = C_i + \sum_{1}^{i-1} \lceil t/T_k \rceil * C_k \tag{3}$$

$w_i(t)$ is the time demand function of i^{th} task when it is released at the critical instant. As can be seen, the tasks from starting to $i-1$ only contribute to this value function. The tasks after the i^{th} task cannot affect as they have lower priority (as task set is sorted). After this calculation, the schedulabilty bound was given as

$$w_i(t) < t \tag{4}$$

Where $w_i(t)$ can be interpreted as demand and the time t can be seen as the supply. So, the demand must be less than the supply for the task set to be feasible. Equation (4) involves checking for time instances the demand of a given task. The time instances that must be checked relates to the the tasks having higher priority than the current. As mentioned earlier, property of RMA asserts that only higher priority tasks can affect the current task. Thus only those time instances need to be checked that are multiples of period of all the high priority tasks.

$$t = j * T_k; \tag{5a}$$

$$k = 1, 2, ..., i; \tag{5b}$$

$$j = 1, 2, ..., \lceil T_i/T_k \rceil \tag{5c}$$

Combining Equation (3) and (5a, 5b, 5c) an algorithm can be implemented using Equation (4) as the checking condition. We shall refer this algorithm to as Time Demand Analysis (TDA)

In this paper we explore a new method to approach the task set to determine schedulability. The main idea in this paper can be applied to any static priority algorithm but for being in synchronization with the previous results we use RMA. Simulation shows our results are better that the other works one in the same field.

In the next section we explore the existing exact schedulability algorithm and propose our improvement.

3. TIME DEMAND ANALYSIS (TDA)

Before presenting the improved algorithm we first describe the TDA in detail. We then implement the TDA so as to provide a common ground upon which the improved algorithm can be compared. As presented above, TDA can be described by three Equations (3-5). We now present how to carry out TDA on a sample task set.

3.1. Implementation of TDA

The first requirement is the sorting of the task sets. The task sets are sorted according to their periods. After the sorting is done, starting from the first task, the computations are carried out to determine the time demand function

value for each task using Equations (3) and (5a, 5b, 5c). If the value of the time demand function for a task does not satisfy Equation (4) then it is deemed as unschedulable. We can divide the algorithm into three phases for better

Algorithm 1: TDA

Input: $task - set$
Output: schedulable or not
Sort the task-set;
repeat
  Calculate time points from Equation (5);
  Calculate $w(t)$ from Equation (3);
  if $w(t) \geq t$ **then**
    return $Notschedulable$;
    exit algorithm;
until *all tasks in the sorted set have been processed*;
return $schedulable$;

understanding:

1. *Determining the Order of execution* - The task-set is sorted as per increasing period. The sorted task sets are sequentially processed one by one. Every task is then subjected to TDA. The task, if any, at which the TDA gives result as unschedulable, is the threshold task. Since the task set is sorted, any task after that also will be unsceduable.
2. *Determining the Discrete time points* - Schedulability test, as mentioned earlier need only be performed at certain time intervals. Using Equation (5) those time points are calculated for every task. These are the points where a instant of one of the task occurs and thus needs to be checked for schedulability.
3. *Computing the time demand function value* - The value calculated from Equation (3) can be computed after all the time points are calculated. At every time point this value is computed and continually summed over. The final sum is the required value that is compared in Equation (4). This value is then checked as per the Equation (4) which gives the decision whether schedulable or not.

The steps are described in an algorithm in Algorithm 1: TDA

Algorithm 2: Sort

Input: $task - set$
Output: sorted $task - set$
for *every task* T_i **do**
  val = period(T_i);
  s_1 = Index of T_i;
  s_2 = 0;
  for *every task* T_k *in* T_{i+1} *to* T_n **do**
    if $val \geq period(T_k)$ **then**
      store index k in s_2;
      val = period(T_k);
  Swap values at index s_1 and s_2;
return sorted $task - set$;

The sorting of the task set can be done in any way. We used associative selection sort [11] where the task set was sorted according to increasing periods. The sorting algorithm is given in Algorithm 2: Sort.

We implemented TDA using C++ with STL on a Intel Core i7 3632Q 2.20 GHz processor. We have measured the performance in terms of iterations so as to provide an uniform performance criteria across multiple types of processors. The worst case of the algorithm occurs when a given task is schedulable as that would require the full task set to be tested. For measuring the performance of the algorithm the task set is being generated randomly. The generation is such that the total utilization of the tasks does not exceed 1.0 but is very near to it. In other words, all the

tasks satisfy the necessary *schedulability bound 1* but do not satisfy the sufficient *schedulability bound 2*. This enables the task set to be processed by TDA. This generation creates equally weighted task sets towards the total utilization. This idea is broadly taken from [12] who gave a detailed method to generate properly distributed random task sets. The task set we have used is not specifically evenly distributed but serves the purpose of implementing the algorithm. An example task set is given in Table 1. The total utilization in this case is 0.8902 which is clearly greater than the *schedulability bound 2* which is 0.717.

Table 1. Sample task set. Total utilization is such that the task set has to be processes by TDA

Period	Execution Time	Utilization (C/P)
62	628	0.098726
55	558	0.098566
94	946	0.099365
60	610	0.098360
51	513	0.099415
75	756	0.099206
90	910	0.098901
62	627	0.098883
81	820	0.098780

This implementation shows that the number of iterations increases at a pseudo-polynomial rate with the number of tasks in the task set. The result is shown in Table 1. We now discuss how can this result be improved.

Table 2. The Performance of TDA measured in terms of iterations with respect to the number of tasks in the task set.

No of Tasks	Iterations
10	20
20	65
30	97
40	185
50	292
100	1239
200	5904
300	14695
400	31189
500	58209

3.2. Scope for improvement

From Table 1 it can be seen that the rate of increase in the number of iterations is pseudo polynomial. The primary drawback of TDA is that the task-set is processed linearly. From one task to the other, every task is visited one by one. In the worst case, when the task set is schedulable, TDA needs to visit all the tasks. This would result in a time complexity of $\theta(n)$ for the outer loop. The selection of tasks results in a linear complexity. We propose an improvement to this approach by not using the linear approach. TDA can also be improved in other ways such as reducing the number of time points in Equations (5a, 5b, 5c) as proposed in [6] and further improved in [7]. Our approach is different from theirs in the way that we reduce the number of tasks to be checked rather than reducing the number of scheduling points for each task.

The rest of the paper discusses our proposed algorithm.

4. IMPROVED TIME DEMAND ANALYSIS

Before looking at the proposed algorithm, one important property of RMA must be established as given below:

A task's schedulability is only affected by those tasks which are having higher priority than it.

This can be seen as a derivation from the nature of RMA itself. Since high priority is assigned to lower period, a task having higher period cannot affect the one having lower period. In TDA, the first step is to sort the task set.

Any task T_i in the sorted task set $T(T_1, ..., T_n)$ will have tasks with lower period before it and higher period after it. If a task is deemed to be schedulable then it is trivial that the task before it must also have been schedulable. This property enables us to deviate from the linear nature of traversal. *Instead of proceeding in a linear fashion we can use a different approach of traversing the task set to determine unSchedulability*. This is the central idea of the improved algorithm. Below we describe in detail the proposed algorithm which we have decided to name as Improved Time Demand Analysis (ITDA).

4.1. Selection Function

We now describe a new way to select a task instead of linear selection as it is done in TDA. Let T_1 be the first task in the sorted set. As per TDA, this task is determined to be schedulable or not by calculating the time points and checking the condition. Suppose this task comes out to be schedulable. For the next iteration, instead of checking the next task, a different tasks T_k is chosen. Now two possibilities can occur as given below,

1. T_k *is not schedulable*: If T_k is not schedulable then the tasks after that can not be schedulable also. In this case the algorithm returns false and exits.
2. T_k *is schedulable*: If T_k is schedulable then, as stated above, the tasks from T_1 to T_{k-1} are also schedulable. The algorithm need not check those tasks. We can proceed on to the next task.

Now we describe in detail how to select the next task. Let n be the total number of tasks present. Let k be the task position of the current task. Let the computed value of time demand function for the current task be $w(t)$ and the corresponding maximum time among the time-points for the task be t. Then we define a ratio r to be

$$r = t/w(t) \tag{6}$$

This ratio is the ratio of $t/w(t)$ from Equation (4). It can be seen that that for a schedulable task, this ratio will be greater than 1. As the task becomes unschedulable, the ratio will go below 1. Thus the criteria that a given task is schedulable can be rewritten as,

$$r \geq 1 \tag{7}$$

To provide an idea of the value of r, a graph is plotted for a given task set for the *ratio* vs *the number of sorted tasks* and is given in Figure 1. As we can see, the ratio converges to 1 as the iteration nears the unschedulable task in the task

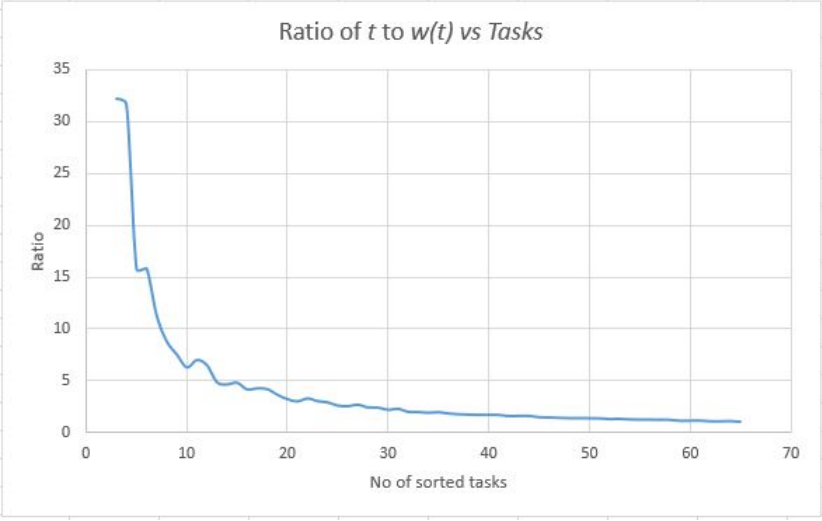


Figure 1. Nature of change of ratio with the tasks in the sorted set. The ratio converges to 1 as the tasks reach uncscheduabe utilization.

set. We now develop a *selection function* that gives an estimate as to how close the ratio is to 1. If it is very close then the next task that is to be selected is near to the current one and vice-verse. This approximation function is a mapping of range of ratio r to the range of task-set. Thus it gives a new position with respect to the position of the current task. As a function it can be represented as

$$m = k + \lceil r * (n/MAX) \rceil \tag{8}$$

Where MAX is the maximum value of the ratio, which is the ratio of the first task's time demand function to time point for that task and m is the next task that is to be executed. So after the computation of T_k the next task to be processed is T_m instead of T_{k+1}. Taking into consideration the selection function we propose our algorithm, ITDA.

Algorithm 3: ITDA

Input: $task - set$
Output: schedulable or Unschedulable

```
Sort the task-set;
k = 1;
repeat
    Task considered k;
    Calculate time points from Equation (5);
    Calculate w(t) from Equation (3);
    if w(t) ≥ t then
        return Notschedulable;
        exit algorithm;
    Calculate ratio r using Equation (6);
    Calculate value m using Equation (8);
    k = m;
until last task is not checked OR ratio ≥ 1;
return schedulable;
```

4.2. Implementation and Results

The algorithm was implemented in the same way as the TDA. Instead of a central linear loop, a loop based on m and $ratior$ is iterated. At each iteration the new value k is calculated which becomes the next task to be scheduled. This decreases the $\theta(n)$ complexity by exponential times. The complexity becomes $\theta(logn)$. The loop iterates until the last task has been checked or the value of ratio goes below 1. On running the algorithm on the same sample task sets as TDA, we obtain the result shown in Table 3. Before discussing the observations in Table 3, in the next

Table 3. The Performance of ITDA comparing the number of iterations with TDA on common task set

No of Tasks	Iterations in TDA	Iterations in ITDA
10	20	8
20	65	11
30	97	29
40	185	45
50	292	57
100	1239	361
200	5904	1365
300	14695	5964
400	31189	11086
500	58209	33609

subsection, we propose an extension to ITDA that enables it to work for multiprocessor environment.

4.3. ITDA for Multiprocessor

To the best our knowledge, there exist no algorithm for schedulability analysis of multiprocessor real-time systems using RMA. In this section, we have extended ITDA to multiprocessor systems. We named this proposed algorithm as Improved Time Demand Analysis for Multiprocessor environment (MITDA). The aim of MITDA is to determine whether a task-set is schedulable in a system having more than one processor available for the processing of the task. We now describe MITA in detail.

MITDA takes input the number of processors available in the multiprocessor environment. In the next step, MITDA divides the task set among the specified number of processors. After the division is done, the algorithm checks the feasibility of tasks in each allocated processor. The steps of the algorithm is described below,

Table 4. The Performance of MITDA on a 4 processor system showing the number of tasks allocated to each of the processor and the total number of iterations taken to carry out the test.

	Number of tasks in	Number of tasks in	Number of tasks in	Number of tasks in	
No of Tasks	Processor 0	Processor 1	Processor 2	Processor 3	Total Iterations
100	16	13	16	14	70
120	18	19	18	16	101
140	21	22	22	21	90
160	24	23	24	23	98
180	28	31	28	27	163
200	33	33	31	33	362
220	36	36	35	32	447
240	38	37	36	39	448
260	39	41	40	43	540
280	46	45	45	44	607
300	50	49	51	47	793

1. *Allocate the tasks to processors*: The tasks are divided among the processors. This division is done by balancing the utilization using Utilization Balancing Algorithm [13].
2. *Determine schedulability of each processor*: Each processor has a number of tasks allocated to it which are balanced in terms of total utilization. On each of these task-sets, ITDA is performed. The result obtained gives the schedulability of each processor. Combining the results of the individual processors, the total schedulability can be estimated.

The full algorithm is summarised in Algorithm 4,

Taking the same task-set of variable number of tasks and running it for a system of 4 processors we get the values at Table 4. In the next section we discuss the results of MITDA.

5. RESULT AND DISCUSSION

We now discuss the results provided via Figure 2, Table 2 and Table 3. TDA gave the base number of iterations depicted in Table 2. Our paper then used this to compare the performance of ITDA and MITDA. The results provided in Table 3 and 4 show the number of iterations using ITDA and for multiprocessor environment using MITDA. Table 4 shows the number of tasks allocated to each processor and the total number of iterations taken for each task set. It can be seen that the number of tasks allocated is such that the total utilization of all the processors are balanced. This also makes the number of tasks allocated to each processor to be roughly equal in number, and thus achieving proper load balancing. Also by comparing this with Table 3 we can see that the number of iterations are less than that of uniprocessor. The graph provided in Figure 2 shows this comparison in detail. The number of iterations decrease from uniprocessor to 2 processor environment and then to 4 processor environment. The Figure 2 showed the variation of the iterations from which we can infer that the algorithm takes less number of iterations with increasing number of processors. Thus, ITDA improves the performance by reducing the complexity from linear to logarithmic terms and MITDA provides a suiTable algorithm to determine schedulability in a multiprocessor environment.

5.1. Comparision with related works

There have been various work done on schedulability analysis of various priority based algorithms. The idea of discrete scheduling points was taken into account in [15], although they proposed it for EDF. Their work was improved and extended in [14]. Recently [7] proposed an Enhanced Time Demand analysis (ETDA) which improved upon Hyper-planes Exact Test proposed by Bini and Buttazzo in [6]. Their implementation decreased the actual number of scheduling points. They measured the performance in terms of total CPU time taken for a given task set to be analyzed. Our alogorithm, ITDA, reduced the number of total tasks considered. Table 5 shows the comparision between ETDA and ITDA. Since the task-set was is randomly generated, we took average of multiple readings to remove any ambiguity.

Algorithm 4: MITDA

Input: $task - set, processors$
Output: schedulable or Unschedulable

```
Sort the task-set;
k = 1;
Initialize utilization = 0 for each processor;
repeat
    Find processor P_k with min utilization;
    Assign next task T_i to P_k;
    Update utilization of P_k += utilization of T_k;
until all tasks have been allocated;
for each processor P_k do
    Considered allocated task-set for P_k;
    k = 1;
    repeat
        Task considered k;
        Calculate time points from Equation (5);
        Calculate w(t) from Equation (3);
        if w(t) ≥ t then
            return Notschedulable;
            exit algorithm;
        Calculate ratio r using Equation (6);
        Calculate value m using Equation (8);
        k = m;
    until last task is not checked OR ratio ≥ 1;
return schedulable;
```

Table 5. Comparision of ETDA [7] with ITDA

No of Tasks	CPU time for ETDA (ms)	CPU time for ITDA (ms)
100	45	32
200	56	41
300	70	54
400	121	76
500	158	97

For the best of our knowledge there has been no multiprocessor implementation of ETDA. Thus MITDA is presented as a new algorithm. Performance of MITDA depends of the efficiency of ITDA. As ITDA is shown to perform better than ETDA, we can say that MITDA is also efficient. The ETDA and the ITDA can be fused together to provide even better results. We carried out our tests independently as that enables us to compare performance with TDA directly.

6. CONCLUSION

We have developed an algorithm that improved the existing TDA as an exact feasibility test for static priority scheduling. We tested the algorithm for RMA. The proposed algorithm is named as ITDA. Further we extended this algorithm multiprocessor environment. We named this algorithm as MITDA. Our algorithm is taking less number of iterations than the existing TDA algorithm. ITDA can be used along with other exact sceduability tests to give even better performance. Further using MITDA we show that on increasing the number of processors the total number of iterations decrease. Our future work will be to introduce task dependency for MITDA. In our current task-sets the tasks are assumed to be independent of each other. Another future work is to execute MITDA in parallel so as to further reduce the time required to compute feasibility. This can be achieved using native pthreads or OpenMP using shared memory architecture.

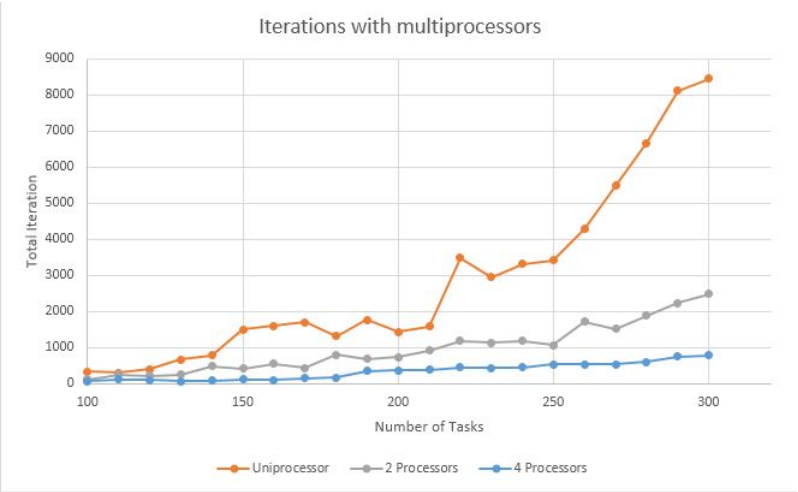


Figure 2. Comparison of iterations with number of processors on running MITDA on diferent types multiprocessor systems. The uniprocessor takes the maximum number of iterations and the iteration decreases with increase in number of processors.

REFERENCES

[1] C. L. Liu and J.W. Layland "Scheduling Algorithms for Multi-Programming in a Hard Real-Time Environment, *Journal of the Assn of Computing Machinery (ACM)* 20, 1, 40-61 January, (1973)

[2] Lehoczky, John, Lui Sha, and Ye Ding. "The rate monotonic scheduling algorithm: Exact characterization and average case behavior." *Real Time Systems Symposium* (1989)

[3] Seto, D., J. P. Lehoczky, L. Sha, and K. G. Shin, On Task Schedulability in Real-Time Control System, *Proceedings of 17th Real-Time Systems Symposium*, pp. 13-21, December, (1996)

[4] T. W. Kuo, Y. H. Liu, and K. J. Lin, Efficient On-Line Schedulability Tests for Priority Driven Real-Time Systems, *Proceedings of the IEEE Symposium on Real- Time Technology and Applications, Washington D.C., USA*, June (2000).

[5] Audsley, Neil, et al. "Applying new scheduling theory to static priority pre-emptive scheduling." *Software Engineering Journal 8.5* (1993): 284-292.

[6] Bini, Enrico, and Giorgio Buttazzo. "A hyperbolic bound for the rate monotonic algorithm." *Real-Time Systems, 13th Euromicro Conference on, 2001.. IEEE*, (2001).

[7] Allah, Nasro Min, and S. Islam and Wang YongJi. "Enhanced Time Demand Analysis." *World Applied Sciences Journal 9.1* (2007)

[8] Manabe, Yoshifumi, and Shigemi Aoyagi. "A feasibility decision algorithm for rate monotonic and deadline monotonic scheduling." *Real-Time Systems 14.2* pp. 171-181 (1998)

[9] Bini, Enrico, and Giorgio C. Buttazzo, "Schedulability analysis of periodic fixed priority systems." *IEEE Transactions on Computers 53.11* pp. 1462-1473, (2004).

[10] Bini, Enrico, and Giorgio C. Buttazzo. "The space of rate monotonic schedulability." *In Real-Time Systems Symposium, 2002. RTSS 2002. 23rd IEEE, pp. 169-178. IEEE*, (2002).

[11] Thomas H.. Cormen, Charles Eric Leiserson, Ronald L. Rivest, and Clifford Stein. "Introduction to algorithms". *Vol. 6. Cambridge: MIT press*, (2001).

[12] Enrico Bini and Giorgio C. Buttazzo. "Measuring the Performance of Schedulability Tests", *Springer, Real-Time Systems, Volume 30, Issue 1, pp. 122 − 154*, May (2005)

[13] Rajib Mall. "Real Time System: Theory and Practice" *3rd Edition, Pearson Education* pp 390-395, (1986)

[14] Lalatendu Behera and Durga Prasad Mohapatra, "Schedulability Analysis of Task Scheduling in Multiprocessor Real-Time Systems Using EDF Algorithm", *International Conference on Computer Communication and Informatics*, Jan. (2012)

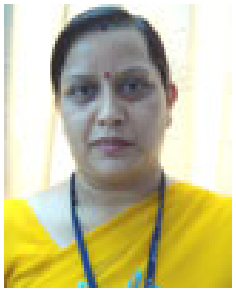
[15] F. Zhang and A. Burns. "Schedulability Analysis for Real-Time Systems with EDF scheduling" , *IEEE Transaction on computers, vol. 58, no. 9*, pp. 1250-1258, September (2009)

[16] Sjodin, Mikael, and Hans Hansson. "Improved response-time analysis calculations." *Real-Time Systems Symposium, 1998. Proceedings., The 19th IEEE*, (1998).

[17] Joseph, Mathai, and Paritosh Pandya. "Finding response times in a real-time system." *The Computer Journal*

29.5, pp. 390-395, (1986)

BIOGRAPHIES OF AUTHORS



Leena Das is Faculty Member and Asst.Professor, School of Computer Engineering, KIIT University, India.

Sourav Mohapatra is a Computer Science and Engineering final year undergraduate student at National Institute of Technology, Rourkela. He is doing his research project on real-time system schedulability analysis under Prof Durga Prasad Mohapatra. He completed a research internship at Indian Institute of Technology, Hyderabad having worked on operating system and parallel programming and has since been working on real-time systems.

Dr. Durga Prasad Mohapatra completed his PhD from Indian Institute of Technology, Kharagpur in 2005 and has since been a professor in National Institute of Technology. He is currently the Head of Department, Computer Science and Engineering. He is a member of IEEE technical society.