ARTICLE IN PRESS

J. Parallel Distrib. Comput. ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at SciVerse ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Lowest priority first based feasibility analysis of real-time systems

Nasro Min-Allah [a,1], Samee U. Khan [b,*], Xiuli Wang [c], Albert Y. Zomaya [d]

[a] Department of Computer Sciences, COMSATS Institute of Information Technology, Islamabad, Pakistan

[b] Department of Electrical and Computer Engineering, North Dakota State University, Fargo, ND 58108-6050, USA

[c] Central University of Finance and Economics, Beijing 100081, PR China

[d] University of Sydney, Sydney NSW 2006, Australia

HIGHLIGHTS

- A feasibility test for RM is proposed by testing system infeasibility.
- Results are applicable to systems where deadlines are not larger than task periods.
- Theoretical foundation for lowest priority task first (LPF) approach is provided.
- The timing constraints of system under investigation are kept intact.
- The presented work dominates existing counterparts from performance perspectives.

ARTICLE INFO

Article history:
Received 9 February 2012
Received in revised form
5 March 2013
Accepted 31 March 2013
Available online xxxx

Keywords:
Feasibility analysis
Fixed-priority scheduling
Online schedulability
Real-time systems

ABSTRACT

The feasibility problem of periodic tasks under fixed priority systems has always been a critical research issue in real-time systems and a number of feasibility tests have been proposed to guarantee the timing requirements of real-time systems. These tests can be broadly classified into: (a) inexact and (b) exact tests. The inexact tests are applied to the task sets that present lower utilization, while the exact tests become inevitable when system utilization is high. The exact tests can be further classified into: (a) Scheduling Points Tests (SPT) and (b) Response Time Tests (RTT). The SPT analyze task set feasibility at the arrival times while the RTT utilize fixed-point techniques to determine task feasibility. All of the available exact feasibility tests, whichever class it belongs to, share pseudo-polynomial complexity. Therefore, the aforementioned tests become impractical for online systems. Currently, both SPT and RTT employ the Highest Priority First (HPF) approach, which determines the system feasibility by testing the schedulability of individual tasks in the decreasing order of priority. In contrast, this work exploits the Lowest Priority First (LPF) alternative which is an aggressive solution based on the observation that the system infeasibility is primarily due to the lower priority tasks and not because of the higher priority tasks. For the average case analysis, our technique demonstrates promising results. Moreover, in the worst case scenario our solution is no inferior to the existing state of the art alternatives. We compare our proposed technique with the existing tests: (a) by counting the number of scheduling points used by a test that belongs to the SPT, (b) by counting the number of inner-most loops executed by an algorithm for the RTT, and (c) by measuring the actual running time of the existing alternatives.

1. Introduction

A real-time system (RTS) is a system where the timing constraints are vital to performing the assigned tasks [34,23,22]. The two main classes of real-time systems are (a) hard and (b) soft real-time systems. Meeting deadlines is absolutely necessary for hard real-time systems. Embedded systems are often hard real-time systems. However, in soft real-time systems there is some room for lateness and a delayed process may not cause an entire system failure. Instead, it may affect the quality of the process or system. Typically, real-time systems are built from concurrent programs, called tasks [18].

There exist a number of task models for real-time systems [14], but in a simple periodic model of hard real-time systems, a task τ_i is represented by the following parameters:

- A task period p_i, which is the interval between two instances (or jobs) of τ_i.

* Corresponding author.
E-mail addresses: nasar@comsats.edu.pk (N. Min-Allah), samee.khan@gmail.com, samee.khan@ndsu.edu (S.U. Khan), xiuli@ios.ac.cn (X. Wang), albert.zomaya@sydney.edu.au (A.Y. Zomaya).

[1] A portion of the work was done at Massachusetts Institute of Technology (MIT) while Nasro Min-Allah was a Visiting Scientist at CSAIL MIT.

0743-7315/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jpdc.2013.03.016

- A worst case execution time c_i.
- Relative deadline d_i, which is measured from the release time.

Let $\Gamma = \{\tau_1, \ldots, \tau_n\}$ denote a set of independent, preemptable periodic tasks, where each task τ_i generates an instance (or a job) at each multiple of p_i. While in operation, all of the tasks immediately get ready for execution on a single processor system as soon as they are released. Moreover, the deadlines are equal to or less than the periods and we assume that there are n number of priority levels available. Therefore, every task has an associated priority.

A rational approach of designing a periodic task set is that for each task τ_i, the expression $c_i \leq p_i$, must always hold. This is due to the fact that the utilization of more than 100% is impractical under a uniprocessor system for any task model. The set Γ is scheduled on a uniprocessor system according to a predetermined scheduling algorithm. Among the existing scheduling mechanisms, priority driven scheduling is the choice for real-time systems [11,21,22,7,31,15,14,17,20,3,4,30,16] which run the task with the highest priority at all scheduling points. The priority driven scheduling is further categorized into: (a) static priority assignment and (b) dynamic priority assignment. From the utilization perspective, the dynamic priority assignment has advantage over static priority counterpart. However, the static priority assignment outclass the dynamic priority policy when it comes to system predictability. Among the (previously known) scheduling algorithms, the fixed priority scheduling is considered to be the choice of the modern real-time systems because of its simplicity and applicability [5,6] and hence the focus of this paper. The most popular scheduling algorithm under the category of the fixed priority scheduling is the Rate Monotonic (RM) algorithm [23]. The RM scheduling algorithm assigns fixed priorities to all of the tasks on their activation rates (periods). The RM scheduling algorithm is being used widely in real-world systems due to its simplicity and reliability [6,13]. According to RM priority assignment policy, for any two tasks τ_i and τ_j, priorities are assigned to tasks in a simple fashion, $priority(\tau_i) > priority(\tau_j) \Rightarrow p_i < p_j$. If there occur ties, then the ties are broken arbitrarily but in a consistent manner. For each task τ_i, its system utilization is defined by: $u_i = c_i/p_i$. The cumulative utilization $U(n)$ of a periodic task system Γ is defined by:

$$U(n) = \sum_{i=1}^{n} \frac{c_i}{p_i}. \quad (1)$$

The RM approach is optimal for the implicit-deadline model, in which the deadlines coincide with their respective periods. However, for the constrained deadline systems, in which the deadlines are not greater than the periods, an optimal priority ordering has been shown to be that of the Deadline Monotonic (DM) scheduling [21]. In the DM scheduling approach, priorities are assigned to tasks which are inversely proportional to the relative deadlines. The RM and DM approaches exhibit identical properties when the relative deadline of every task is proportional to its corresponding period. Although both the RM and DM techniques can be used (interchangeably) for our constrained-deadline (where every task has its deadline not larger than its period), to align with previous literature, we use RM in this work.

To determine if a given set of periodic tasks meet all of their deadlines, is considered a special case of the validation problem: We are given

- a periodic task set (Γ),
- a constrained deadline task model, and
- an RM scheduling algorithm.

We must utilize the aforementioned items to determine if all of the deadlines d_i of every task τ_i $\forall i$, $1 \leq i \leq n$ will be met on an underlying uniprocessor system. To determine whether a feasible schedule exists for Γ, schedulability tests are performed that fall into three main classes [1,9]: (a) Sufficient Condition: the Γ is definitely schedulable if the schedulability test belongs to this class is satisfied and it is indecisive when the schedulability test fails, (b) Necessary Condition: if the schedulability test is passed then we do not know whether Γ is schedulable or not, however the task set is definitely unschedulable if the test fails, (c) Exact Conditions: are both sufficient and necessary at the same time. In rest of the paper, we use schedulability test and feasibility analysis interchangeably. The feasibility analysis can be performed either offline or online [22]. For offline systems, the computational complexity is not deemed to be a major issue. Therefore, the corresponding algorithms are evaluated from the perspective of the quality of the feasibility tests. Conversely, in online systems, task scheduling is determined on arrival, which means that the corresponding feasibility tests must be performed at run time. Therefore, the online tests must be fast — an online algorithm that takes so long that it leaves insufficient time for the tasks to meet their deadlines is deemed useless [17,33,29,19,26,28,10].

The feasibility of Γ, which has a low system utilization can be determined with the sufficient conditions that are available in the real-time system literature, such as $U(n) \leq \ln(2)$ [22] and/or $\prod_{i=1}^{n}(u_i+1) \leq 2$ [15]. However, the feasibility of Γ having a higher system utilization must be determined by utilizing both the necessary and sufficient conditions. This being the focus of this paper.

The necessary and sufficient conditions determine the RM schedulability of a task on the basis of a task's worst case response time. Two approaches, namely: (a) scheduling point and (b) fixed-point techniques, are used to determine a task's maximum possible response time. However, both of the aforementioned techniques are known to be computationally expensive and are deemed impractical to be used for online analysis. Although the order in which the system feasibility is evaluated does not matter [30], these necessary and sufficient conditions [5,6,23,11,21,14,20,3,4,30,16,8,25,32,33,29,28] traditionally use the Highest Priority First (HPF) approach. It is worth mentioning that the main attraction for testing system feasibility with HPF is to check if the system is feasible and concluding this takes a considerably longer time [22,19,26]. The aforementioned is attributed to the fact that higher priority tasks are always schedulable and finding the infeasible task (normally lower priority task) means evaluating the feasibility of all higher priority tasks before that infeasible task. Guarantee scheduling of higher priority task is due to the implicit characteristics of RM scheduling algorithm because CPU is assigned to higher priority task when the overloading situations occurs. In contrast, the Lowest Priority First (LPF) counterpart evaluate system from infeasibility perspective and hence system feasibility is determined in reverse order i.e., starting with lowest priority. The advantage of LPF over HPF approach is implicitly highlighted in [12,2] where schedulability of the system is addressed through Response Time Tests (RTT) class and results are established to obtain higher initial values for the tasks. However, as per existing literature, no study has been made so far on the feasibility tests belonging to scheduling point class. In this paper, we investigate the feasibility problem by presenting LPF based feasibility test and compare results with both RTT and Scheduling Points Tests (SPT) classes. Moreover, we provide the necessary foundation for the LPF strategy and make explicit analysis of LPF with HPF counterpart. We also must note that the HPF approach is a well respected and widely used technique for real-time systems and other computing systems where the focus is on determining the system feasibility [24,9,22].

Our prior work in [26] explored the concept of composite deadlines for obtaining higher systems utilization by making tasks deadlines in a harmonic fashion. Using the HPF approach, the work presented in [26] guarantees 100% CPU utilization for larger deadlines under dynamic priority scheduling. In [29], we revisited the fixed priority scheduling domain and performed a comparative

Please cite this article in press as: N. Min-Allah, et al., Lowest priority first based feasibility analysis of real-time systems, J. Parallel Distrib. Comput. (2013), http://dx.doi.org/10.1016/j.jpdc.2013.03.016

Table 1
CPU utilization, applications, and recommendations [19].

Utilization (%)	Zone type	Typical applications
26–50	Very safe	Various
51–68	Safe	Various
69	Theoretical limit	Embedded systems
70–82	Questionable	Embedded systems
83–99	Dangerous	Embedded systems
100	Overload	Embedded systems

analysis of feasibility tests. Similarly, a procedure was developed in [28] for obtaining the optimal values for task computation times so that any given performance index such as system utilization or mission life is maximized. By task shifting approach, we identified a suitable systems speed in [27] that enables all the cores to run on a uniform speed and hence reduces the power consumption of the overall system. All of the aforementioned works done are based on the HPF policy. However, when the (real-time or computing) system is viewed from infeasibility perspective, the HPF technique reveals poor performance. This observation suggests to tackle the RM-feasibility problem of unschedulable task sets with the LPF approach that works in an aggressive manner. This LPF approach is under investigation in this paper. The supremacy of LPF over HPF is discussed in detail through both qualitative and quantitative analysis.

It can be observed from Table 1 that when the system utilization is below 69%, the inexact tests are applicable and the whole task set is RM feasible (on uniprocessor systems). On one hand, this predictability is obtained on the price of system utilization. That is to say that 31% system utilization is compromised and hence the system remains underutilized. On the other hand, when the system utilization is higher, only the exact tests can determine the system feasibility and in such cases the systems are generally infeasible as shown in Table 1. Based on the aforementioned observations, in this research, we analyze the task set from the point of view of system infeasibility. Moreover, we also illustrate that the LPF [12,2] approach is more suitable for the aforementioned task model, as it needs to determine the system infeasibility by probing fewer number of tasks compared to the HPF approach. The effectiveness of the LPF approach is measured by recording the number of points where a task's schedulability is tested for the SPTs and the number of the inner-most loops for RTTs, in addition to measuring the actual run times of RTA, RTI and HET. Table 1 summarizes the CPU utilizations and the corresponding usability. The table also reports that the real-time systems depicting high system utilization are more susceptible to deadline misses. Therefore, the authors conclude that such systems would be more appropriate for our study of investigating real-time (or computing) systems from the perspective of system infeasibility.

In this work, the authors propose the following two mechanisms to decide the RM feasibility/infeasibility of a task set:

(a) Faster Feasibility Test: We observed that if there is a common scheduling point for all of the tasks in the systems and if the lowest priority task is RM feasible at that point, then the entire task set is feasible. This is due to the fact that the value of workloads are non-decreasing, as the RM feasibility of tasks in decreasing priority order are considered. Therefore, testing the feasibility of the lowest priority task at a common scheduling point for all of the tasks result in a faster feasibility test.

(b) Faster Infeasibility Test: A task set is RM infeasible when at least one task is RM infeasible. A lower priority task is more vulnerable to a deadline miss than that of a higher priority task because a lower priority task suffers from interference from all the higher priority tasks. Therefore, the Lowest Priority First (LPF) approach to test the RM infeasibility results in faster decisions.

The remainder of this article is organized as follows: a motivational example is described in Section 2 and then in Section 3, we provide the necessary definitions and some background information about the topic. The related work is discussed in Section 4 that encapsulates both SPT and RTT in detail. This is followed by the details of the proposed analysis scheme in Section 5, which is based on the observation that lower priority tasks are more vulnerable to deadline misses than the higher priority tasks. Experiments have been performed and the results are shown for the performance assessment in Section 6, followed by worst/best case analysis for the proposed scheme. Finally, we conclude the work in Section 7.

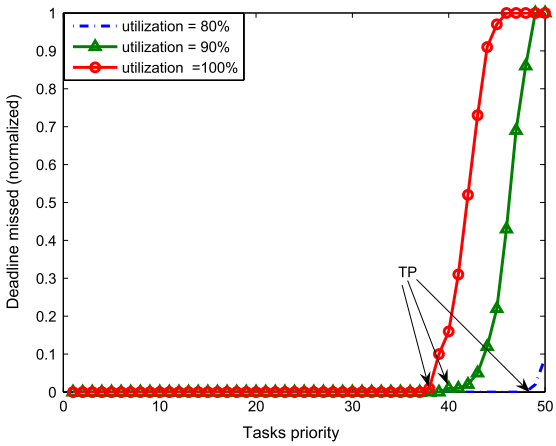


Fig. 1. Non-schedulability of lower priority tasks.

2. Motivational example

As a motivational example, the authors show the infeasibility of the lower priority tasks when the utilization is within the range of [80, 100%]. The RM feasibility of a task set consisting of 50 randomly generated tasks is plotted (in Fig. 1) under varying system utilization (0.8, 0.9 and 1.0). The x-axis represents the decreasing task priorities and the y-axis represents the (normalized) number of missed deadlines. For the experiment, random task periods were generated within the range of 100–100 000 with uniform distribution. For the corresponding task execution demands, again random values were extracted within the range of uniformly distributed values of $[1, p_i]$. The task priorities were assigned according to the RM approach. To deduce enough confidence in our results, each experiment was performed 1000 times. Each point on the plot reflects the average number of deadline missed under the specified utilization. At a utilization lower than 69%, all of the tasks were schedulable under the RM scheduling approach. We must observe that some of the tasks may miss deadlines when the utilization may increase. Interestingly, it was found that the lower priority tasks were more susceptible to the deadline misses among all of the tasks within the task set. In other words, if a task set is deemed infeasible, then it is mainly because of the 'hard to schedule' lower priority tasks. The main reason behind the aforementioned phenomenon is that the tasks are priority scheduled on a uniprocessor system. That is to say that the CPU time is allocated to the higher priority tasks and the rest of the CPU time (if any) may be assigned to the lower priority tasks. As a high utilization is expected, the system may become infeasible because there exists at least one unschedulable task that makes the task set infeasible, which possesses the lower (if not the lowest) priority among all of the tasks. Referring back to Fig. 1, the point that differentiates the schedulable tasks from unschedulable tasks (under a given utilization), is denoted by the Threshold Point (TP). It can be observed that when the system utilization is increased, the TPs arrives early. For the system utilization of 0.8, 0.9, and 1.0, the TPs are 48, 42, and 38, respectively.

Please cite this article in press as: N. Min-Allah, et al., Lowest priority first based feasibility analysis of real-time systems, J. Parallel Distrib. Comput. (2013), http://dx.doi.org/10.1016/j.jpdc.2013.03.016

Table 2
Notations.

Notation	Meaning
Γ	The set of periodic tasks
τ_i	i-th task, $\tau_i \in \Gamma$
c_i	Worst case execution time of τ_i
p_i	Period of τ_i
d_i	Deadline of τ_i
u_i	Utilization of τ_i
n	Number of elements in Γ
$U(n)$	Utilization of Γ
$priority(\tau_i)$	Priority of τ_i
$\mathbf{T_i}$	Subset of task set having a priority higher than τ_i
R_i	Maximum response time of τ_i
S_i	Set of scheduling points for τ_i
$H_i(t)$	Set of scheduling points for τ_i where $H_i(t) \subseteq S_i$
L_i	Schedulability condition for τ_i
L	Feasibility condition for Γ
$\digamma$	Set of scheduling points, $\digamma = \bigcap S_i \neq \phi, \forall i, 1 \leq i \leq n$
$W_i(t)$	The cumulative workload due to τ_i at time t

3. Definitions and background

Before proceeding on a more detailed description, we first introduce notations in Table 2, along with a few definitions that will be used frequently in rest of the paper.

Definition 1 (*Critical Instant [23]*)**.** A critical instant for a periodic task τ_i is the time when τ_i is released simultaneously with request from all of the higher priority tasks.

Definition 2 (*Maximum Response Time [20]*)**.** The longest time ever taken by an instance of the task from its release time until the time it completes its required computation. For a given τ_i, the Maximum Response Time is represented by R_i.

Definition 3. $\mathbf{T_i}$ represents a subset of task set Γ having a priority higher than τ_i.

The authors assume that if the critical instant occurs at $t = 0$, then it has been proven that the first job that belongs to task τ_i has the largest possible response time among all of the jobs [20]. For our task model, at most one such job exists at any instance of time t. Therefore, in the rest of the paper a task τ_i implicitly refers to the first job of a task set. To determine the feasibility of a task τ_i in an interval $[0, d_i]$ (in addition to c_i), the higher priority tasks also must be considered. We can observe that at any instance of time t, there are at least $\lceil \frac{t}{p_j} \rceil, j \in \mathbf{T_i}$ number of iterations of higher priority tasks that occupy $\lceil \frac{t}{p_j} \rceil c_j$ units of processor time. Therefore, a necessary and sufficient condition of task feasibility is: $\exists R_i \leq d_i$ for τ_i, where R_i is equal to the smallest value of t that satisfies the following [20,16]:

$$t = c_i + \sum_{j \in \mathbf{T_i}} \left\lceil \frac{t}{p_j} \right\rceil c_j. \tag{2}$$

To determine that such a t does exist, Eq. (2) is solved by utilizing the highest priority first based approaches discussed in Section 4, in detail. In contrast, to address the same problem, we adopt the lowest priority first approach in Section 5.

4. Highest priority first approach

4.1. Scheduling point tests

There are two major types of scheduling point tests, that are detailed in the following two subsections.

4.1.1. Time demand analysis (TDA [20])

The workload $W_i(t)$ constituted at time t by τ_i consists of its execution demand c_i as well as the interference it encounters due to higher priority tasks. We can represent this phenomenon by the following expression:

$$W_i(t) = c_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{p_j} \right\rceil c_j. \tag{3}$$

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

$$L_i = \min_{0<t\leq d_i} (W_i(t) \leq t). \tag{4}$$

The condition L_i is always satisfied when the cumulative workload due to task τ_i at point t is less than or equal to time t. Interchangeably, we assume that $L_i = 1$, when a task τ_i is schedulable, and 0 otherwise. A task τ_i completes its computational demands in time window $t \in [0, d_i]$, if and only if the request from the $i-1$ higher priority tasks as well as the computational time of τ_i are completed at or before the instance of time t. Because t is a continuous variable, there are infinite numbers of points needed to be tested during determining the task schedulability according to the RM scheduling algorithm. In a classic paper [20], the first attempt was made to restrict the infinite number of scheduling points in the interval $t \in [0, d_i]$. The authors showed that $W_i(t)$ is constant, except at finite number of points. Those points occur when tasks are released within the system, termed the 'RM scheduling points'. Consequently, to determine if τ_i is schedulable or otherwise, $W_i(t)$ is only computed at instances of time that are multiples of $p_i \leq p_j$, $1 \leq j \leq i$. Specifically, let

$$S_i = \{ap_b | b = 1, \ldots, i; a = 1, \ldots, \lfloor p_i/p_b \rfloor\}, \tag{5}$$

where a and b are positive integers. In other words, τ_i is feasible if $L_i = \min_{t\in S_i}(W_i(t)/t \leq 1)$ is satisfied for some point t. Consequently, the entire task set Γ is schedulable for all of the task phasings using a RM approach if and only if:

$$L = \max_{1\leq i\leq n} L_i \leq 1. \tag{6}$$

4.1.2. Hyper-Planes Exact Test (HET)

Refs. [5,6] provide a formulation, acronym HET that reduces the number of scheduling points for τ_i from set L_i to a reduced set $H_i(t)$. For any task τ_i, the HET begins with p_i and expands its search space by:

$$H_i(t) = H_{i-1}\left(\left\lfloor \frac{t}{p_i} \right\rfloor p_i\right) \cup H_{i-1}(t), \tag{7}$$

where $H_0(t) = \{t\}$.

4.2. Response Time Tests (RTT)

Instead of testing feasibility at fixed points, Refs. [3,4,30,16] proposed an iterative method to solve Eq. (2), termed Response Time Analysis (RTA). Because the response time of a job, τ_i, is at least equal to execution time of that job, so its worst case execution time c_i is used as the initial value. Let $R_i^{\#n}$ be the n-th approximation to the exact value of the response time R_i of a task τ_i. In this process, the l-th iteration for $l \geq 1$, the term $R_i^{\#l+1}$ can be calculated as:

$$R_i^0 = c_i, \tag{8}$$

$$R_i^{\#l+1} = c_i + \sum_{j \in \mathbf{T_i}} c_j \left\lceil \frac{R_i^{\#l}}{p_j} \right\rceil. \tag{9}$$

Please cite this article in press as: N. Min-Allah, et al., Lowest priority first based feasibility analysis of real-time systems, J. Parallel Distrib. Comput. (2013), http://dx.doi.org/10.1016/j.jpdc.2013.03.016

It can be seen that the above expression can be easily solved after a few number of iterations. This is due to the fact that the sum is an increasing function of l. There are two cases where the loop is terminated: (a) $R_i^{\#l+1} = R_i^{\#l}$ and (b) $R_i^{\#l} \leq p_i$ for some l or $R_i^{\#l+1} > p_i$, whichever comes first. In the case of (a), τ_i is schedulable, while τ_i is not schedulable in the latter case. Ref. [30] proposes an improvement to RTA, termed Response Time Improved (RTI), which utilizes R_{i-1}^0 for assigning initial values to R_i^0, i.e. $R_i^0 = R_{i-1}^0 + c_i$. Consequently, RTI concludes faster as compared to RTA.

5. Lowest priority first approach

In this section, we present our novel approach for a robust exact test, with an improved performance compared to the existing computationally expensive exact tests. Our proposed technique is inspired by the observation that the lower priority tasks are more susceptible to deadline violations than the higher priority tasks. As discussed earlier, to carry out task set feasibility analysis, individual tasks can be tested in any order. That is, either starting from the highest priority task τ_1 to the lowest priority, or vice versa [30]. Consequently, we prove (in the subsequent text) that it is a reasonable approach to address the validation problem with the LPF technique. Before presenting the main results, we establish a necessary formulation with the help of existing literature for testing feasibility in the reverse order, which lead us to the main contribution of the paper.

Lemma 1. *Given a non-idling scheduling algorithm, a periodic task system Γ is always feasible when $\Gamma = \{\tau_1\}$.*

Proof. If τ_1 is infeasible, then $c_1 > p_1 \Rightarrow u_1 > 1$. As a consequence, the required amount of computational time for a task exceeds the available time, which is impossible. Because the condition $c_i \leq p_i$ always holds for any task τ_i, and when there is only one task in the system, the same task is always schedulable and hence Γ is declared feasible. □

Extending the above lemma, we can show that an infeasible task set $\Gamma = \{\tau_1, \tau_2\}$ consists of a schedulable task τ_1 and an unschedulable task τ_2. Because the schedulability of τ_1 follows from Lemma 1 and hence it is τ_2 that makes Γ infeasible.

Remark 1. Γ is infeasible if there exists an unschedulable task τ_i, $1 \leq i \leq n$.

Although there may exist a RM-infeasible task set even when $U(n) \leq 1$, (for our task model) a task set is always infeasible when $U(n) > 1$. On the other hand, in situations when $U(n) \leq 1$, it can be concluded that when $\Gamma = \{\tau_1, \tau_2, \tau_3\}$, τ_3 has a higher probability to miss its deadline compared to τ_2 and τ_1 as both higher priority tasks are likely to be schedulable even when Γ is infeasible. In the light of the aforementioned discussion (Lemma 1 and Remark 1), one should be concerned with finding the infeasibility of unschedulable low priority tasks and not with the higher priority tasks. In our algorithm, we perform task set feasibility analysis starting with the lowest priority task τ_n and move our way to the highest priority task τ_1. As soon as an unschedulable task τ_i is identified, we immediately conclude that it is unnecessary to test the schedulability of the remaining $(i-1)$ higher priority tasks. This observation results in a simple feasibility analysis that can be used in hard real-time systems due to clear advantages over others. Our experimental results also reverify the efficiency of this approach in Section 6. This alternative (proposed) scheme is much faster than its corresponding exact solutions.

5.1. LPF with scheduling point test approach

In this section, we develop the necessary mathematical formulation for our proposed LPF based approach.

From the relationship existing among the scheduling point sets that for any two sets S_i and S_{i+1}, a scheduling point t in set S_i populated due to τ_i is also a member of the superset S_{i+1}. Because the maximum scheduling point in a set S_i is p_i, and the maximum point in set S_{i+1} is p_{i+1}, there exist a relationship between set S_i and S_{i+1}. As p_i and p_{i+1} are distinct ($p_i < p_{i+1}$) then $S_i \subset S_{i+1}$ holds and hence $p_{i+1} = \max(S_{i+1}) \notin S_i$. It is clear from the above discussion that p_{i+1} is a point in the set S_{i+1}, which is the maximum in the set and this point is missing in S_i because p_{i+1} is due to the task period of τ_{i+1}, which is of course missing in set S_i. This observation also reveals that a point $t \in S_{i+1}$ is not necessarily the member of set S_i, as reported in Eq. (5). However, the converse is not true. Thats is to say that if a point c is a member of a set S_{i+1}, then the same point c must also be a member of set S_i. Referring back to Eq. (5), the task set is infeasible if and only if any task τ_i is unschedulable and from Fig. 1, we can observe that the probability of τ_n to miss the deadline is the highest. Therefore, when started from the LPF approach, if τ_n is unschedulable, then Γ is infeasible. However, if τ_n is schedulable, then we need to test schedulability of $\tau_{n-1}, \tau_{n-2}, \ldots, \tau_1$. The following theorem explains that if: (a) we find some point(s) that is(are) common in all of the sets $S_n, S_{n-1}, \ldots, S_1$ formulated due to the higher priority tasks $\tau_n, \tau_{n-1}, \ldots, \tau_1$ and (b) when τ_n is schedulable at any point(s) in that set, then it is to be concluded that Γ is feasible based on Eq. (6). Consequently, checking the schedulability of other tasks $\tau_{n-1}, \tau_{n-2}, \ldots, \tau_1$ is deemed useless.

Theorem 1. *A task set Γ is always RM-feasible if the lowest priority task τ_n is schedulable at some point $t \in F$ such that $F = \bigcap S_i \neq \phi$, $\forall i$, $1 \leq i \leq n$.*

Proof. Because $F \neq \phi$, we can conclude that there exists at least one scheduling point that is common in all of the RM scheduling point sets. Let x be the common scheduling point. That is to say that the schedulability of every task τ_i has to be tested at x, with HPF approach. We know from previous discussion that a task τ_n is schedulable when

$$W_n(x) = \sum_{j=1}^{n} \left\lceil \frac{x}{p_j} \right\rceil c_j \leq x. \tag{10}$$

The aforementioned is due to the fact that $W_n(x) \geq W_{n-1}(x) \geq \cdots \geq W_1(x)$. Therefore, if Eq. (10) holds, then $W_i(x) \leq x$ must also hold for $i = n-1, n-2, \ldots, 1$. In other words, if x is a feasible point for τ_n, then x also must be a feasible point for τ_{n-1}, $\tau_{n-2}, \ldots, \tau_{n-(n-1)}$. Therefore, the workload presented by any higher priority task in $\mathbf{T_i}$ is also satisfied at point x. This completes the proof. □

If the aforementioned theorem holds, then the schedulability of only one task determines the feasibility of Γ. However, if $F = \phi$, then the feasibility of Γ is inconclusive. In other words, if $F = \phi$ and τ_n is feasible, then we have to test the schedulability of lower priority tasks (in order of decreasing priority). In contrast, irrespective of F, the non-schedulability of τ_n confirms the RM-infeasibility of Γ.

Corollary 1. *If $L_i \leq 1$ and $F = \bigcap S_j \neq \phi$, for $j = 1, \ldots, i$, then all $i-1$ higher priority tasks are schedulable*

Proof. Follows from Theorem 1.

It can be seen that under the ideal cases, only one task is sufficient to conclude the whole set feasibility. If the test fails for τ_n, then it goes to τ_{n-1} and so on and so forth in descending priority order as mandated in Corollary 1. Therefore, the LPF approach

Please cite this article in press as: N. Min-Allah, et al., Lowest priority first based feasibility analysis of real-time systems, J. Parallel Distrib. Comput. (2013), http://dx.doi.org/10.1016/j.jpdc.2013.03.016

analyzes system feasibility much more quickly, as the schedulable point (the point at which all higher priority tasks are schedulable) appears quickly and the test determines the feasibility much earlier as compared to the HPF counterpart. Similarly, an infeasible task is identified much earlier and the remaining higher priority tasks are skipped in the process. Our proposed technique is documented as Algorithm 1. □

Algorithm 1 LPF with SPT

```
procedure LPF(Γ)
for all τ_i ∈ Γ : Γ is in non-increasing priority order do
  if (L_i ≤ 1) then
    τ_i is schedulable;
  end if
  if (Γ ≠ φ) and (t ∈ S_i) and (L_i ≤ 1) then
    Γ is feasible;
    break;
  end if
end for
if (L ≤ 1) then
  Γ is feasible;
else
  Γ is infeasible;
end if
end procedure
```

As established in the real-time systems literature [5,6], the SPT based approaches allow real-time system designers to fine-tune their systems by adjusting the task parameters. Moreover, the formalization of the SPT tests allows the researchers to further investigate the feasibility problem by testing a reduced number of scheduling points [5,6]. The most appropriate criterion for feasibility test that belongs to SPT based approaches is the number of scheduling points—the lower the number of scheduling points utilized, the better is the test. Based on the aforementioned discussion, the effectiveness of the proposed tests is compared with the existing solutions in the subsequent text.

5.2. LPF with fixed point iteration

The fixed point iteration tests are more efficient compared to the scheduling point tests as discussed in Section 4. The fixed point iteration tests require a single loop to determine the task feasibility. As far as the fixed-point iteration schemes are concerned, the RTI approach concludes considerably faster compared to the RTA approach. Therefore, the RTI approach should be the choice for an LPF based technique. However, for any τ_i, the RTI approach needs to start R_i^0 with R_{i-1}—the response time of tasks τ_{i-1} must be known, which is only possible when the task feasibilities are evaluated in the descending priority order. Therefore, the RTI approach is inapplicable (at least for our case). As a consequence, we opt for the RTA approach because its initial guess is the execution demand of the current task. Taking this guess value is understandable and cannot violate the feasibility result if considered in the ascending priority order. Consequently, the initial value for τ_i is determined to be $R_i^0 = \sum_{j=1}^{i} c_j$. With the aforementioned limitation, our solution simply inherits the RTA concept and comparisons with existing tests, such as TDA, HET, RTA, and RTI are made in Section 6.

6. Experimental results and analysis

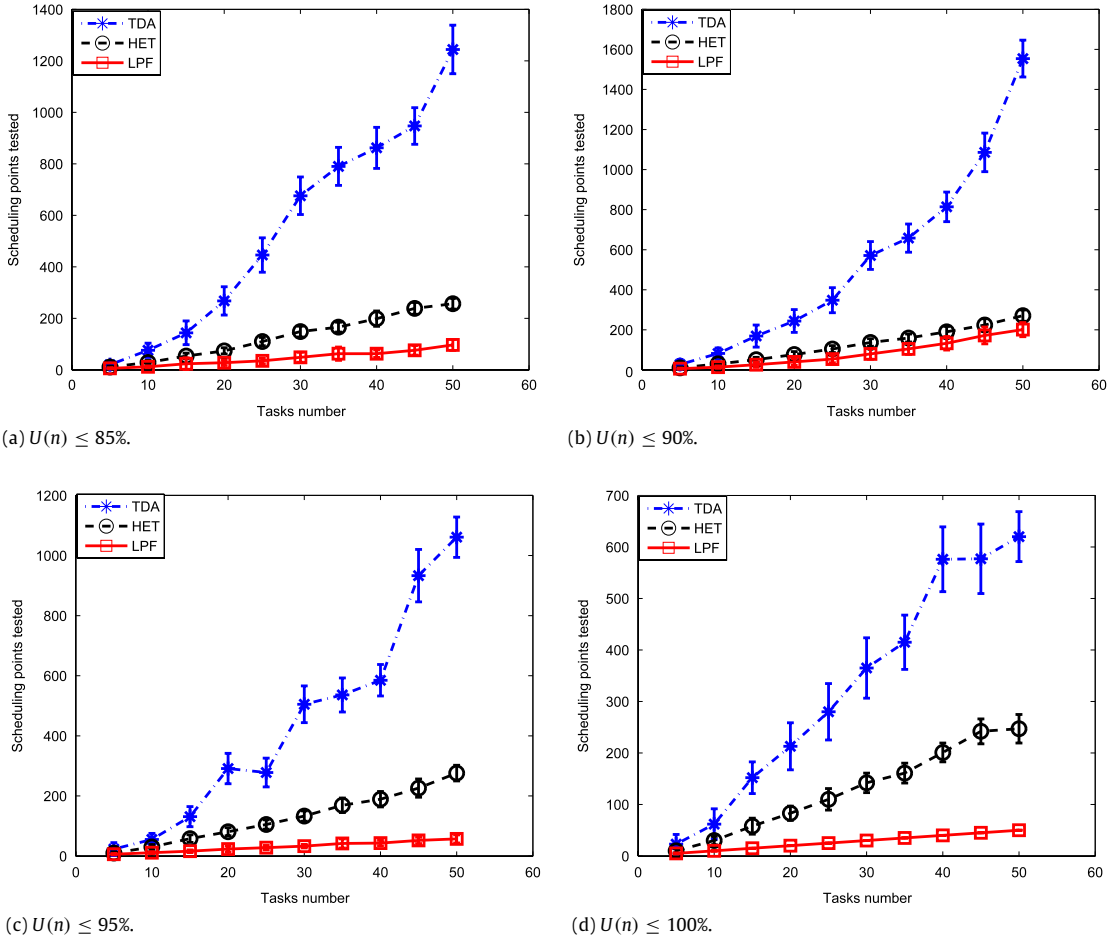
In this section, we compare our proposed technique with the existing exact feasibility analysis techniques. The task parameters, such as computation times c_i and task periods p_i of an individual task τ_i are obtained with the standard practices in real-time systems [5,6]. We implicitly assume that the task deadlines are in agreement with the task periods. For the task periods, we generate random values within the range of 100–100 000 with uniform distribution. For the corresponding c_i of task τ_i, random values are taken from the range of uniformly distributed values of $[1, p_i]$. The RM policy is adapted for assigning priorities to the individual tasks. We evaluate the performance based on three established criteria [5,6,12]: (a) the number of RM scheduling points used, (b) the number of inner-most loops executed, and (c) the actual run times. Criterion (a) is used for comparing the test in the SPT class, criterion (b) is used for measuring the performance of tests in the RTT class, and (c) is used for comparing the fastest solutions existing in literature, namely, RTA/RTI. Because we are focusing on finding the infeasibility of a given task set, we keep $\ln(2) \leq U(n) \leq 1$.

It is an established fact that a task system with $U(n) \leq \ln(2)$ is always feasible and it can be easily determined with simple feasibility tests, such as $U(n) \leq \ln(2)$ [22] or $\prod_{i=1}^{n}(u_i+1) \leq 2$ [15]. On the other hand, as pointed out in Fig. 1, the lower priority tasks start missing the deadlines when the system utilization is increased. For task set generation, the same criteria that is used in Section 2, is adapted here. According to our task creation criterion, the tasks start missing the deadlines with higher $U(n)$. Therefore, we keep the system utilization at 85%, 90%, 95%, and 100% for evaluating the feasibility tests discussed in the Sections 5.1 and 5.2.

6.1. Number of scheduling points

Here, we demonstrate the effectiveness of the LPF based approach in terms of reducing the number of scheduling points. Because one aspect of the current research is on reducing the number of scheduling points, the aforementioned becomes an appropriate criterion to evaluate the performance of a feasibility test lies within the SPT class. Fig. 2 reports the comparison among the tests (TDA, HET, and LPF) by counting the number of points that are actually utilized before the corresponding techniques conclude the system feasibility. The plots in Fig. 2 were obtained by generating a series of periodic tasks with varying task set sizes ranging from 5 to 50 with an increment of 5 tasks. Each point within a plot represents the average value of a total of 2000 runs. The points represent the average values after 2000 runs while the bars show the standard deviations. When $U(n) = 85\%$–90%, it is worth mentioning that the majority of the task sets generated at this utilization remain feasible. Though ln(2) is a theoretical limit of task system feasibility, it has been reported that the system with utilization less than or equal to 85% are generally feasible [20]. The system utilization of (85%) appears the most demanding ones for the LPF based approach because all of the n tasks must be evaluated before a conclusive decision. Because the utilization (85%) is not very high, the execution demands of the individual tasks are quite descent. This descent system utilization of an individual task means that the task τ_i is feasible within the interval $[0, t_i]$, $t_i \ll p_i$. As a result, fewer points are needed to be tested for the TDA based approach. However, the LPF based approach encounters a larger set Γ. Therefore, the probability of scheduling τ_n at some point x is quite higher. As a result, fewer points suffice to determine the system feasibility. As the system utilization increases, the number of the required test points also decreases (see Fig. 2(c) and (d)). The aforementioned is due to the fact that a task τ_i is infeasible and hence a conclusion is made early. For the plots, we observe that the HET based approach reduces the most number of scheduling test points ($H_i \subseteq S_i$) as compared to TDA. Both the TDA and LPF based approaches need to test the system feasibility based on a predefined set that primarily depends on the task periods as reported in Eq. (5). Moreover, it also can be observed from Fig. 2 that the maximum number of loops occurs when system utilization is 85%–90% and it is understood as all of the tasks are deemed feasible within this range. For the LPF based

Please cite this article in press as: N. Min-Allah, et al., Lowest priority first based feasibility analysis of real-time systems, J. Parallel Distrib. Comput. (2013), http://dx.doi.org/10.1016/j.jpdc.2013.03.016



(a) $U(n) \leq 85\%$. (b) $U(n) \leq 90\%$.

(c) $U(n) \leq 95\%$. (d) $U(n) \leq 100\%$.

Fig. 2. Reduction in average number of scheduling points.

approach, the cumulative effect is high as the algorithm starts in the reverse order. Therefore, it must test the RM feasibility of more tasks—up to n in the worst case scenario. Even then, for the RM schedulability of the systems, there are high chances of getting a point $x \in \Gamma$, such that $W_i(x) \leq x$, which makes testing the schedulability of $\mathbf{T_i}$ unnecessary. When applied, LPF can avoid such cases by concluding system infeasibility much early and hence utilizes a reduced number of scheduling points.

In contrast, the aforementioned situation becomes different at higher system utilization. It can be observed from Fig. 2(c) and (d) that the LPF based approach returns the decision on the task feasibility much faster compared to the rest of the techniques. This benefit in the favor of LPF is due to the fact that the task set infeasibility for all sizes of the task set is determined much early due to its implicit nature of exploiting reverse priority order. The LPF becomes more superior when n is large. As discussed above, there is no rationale in testing the remaining $(n-1)$ tasks when τ_n is identified as RM unschedulable. From the aforementioned discussion, we may also conclude that the LPF based approach is more effective when applied to a large task set or when the underlying system utilization is quite high.

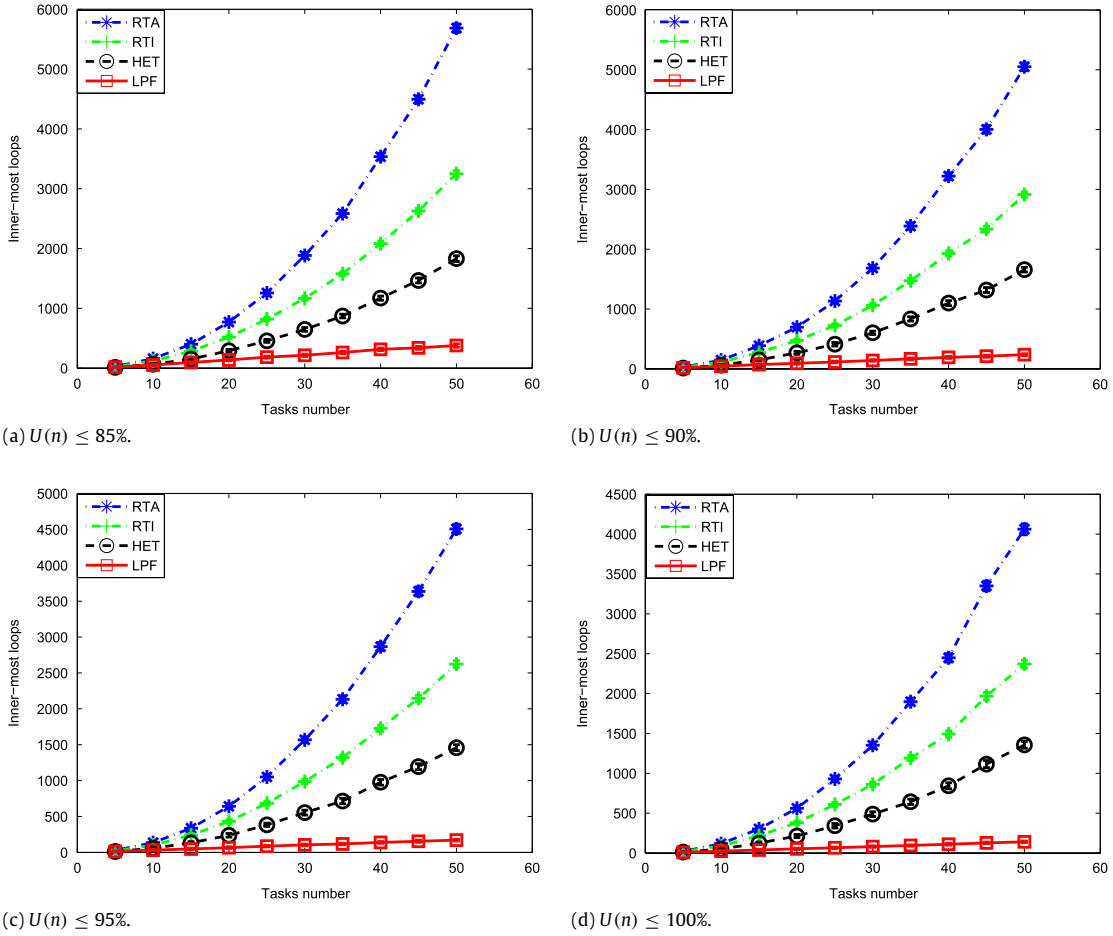
6.2. Number of inner-most loops

As mentioned in [5,6], this criterion (enumerating the number of calls to the inner-most loops) is well suited for comparing feasibility tests based on response time, such as RTA, RTI, HET and LPF. These techniques provide solution after a finite number of iterations [5,6]. Fig. 3 illustrates the results of our second experiment that compares LPF with: (a) RTA, (b) RTI, and (c) HET approaches. The point in a plot is an average of 2000 runs while respective bars reflect standard deviations. The computational complexity of the tests classified as RTT are (usually) measured by recording the number of calls to the inner-most loops. Based on the aforementioned criterion, if a test makes a fewer number of calls to the inner-most loops, the better the test would be, and vice versa. The advantage of HET over RTA and RTI is reported in [5,6]. Recall that the RTA is overshadowed by the performance of the RTI based approach because of getting higher values as starting point to determine feasibility. Average values are plotted in Fig. 3, after 2000 runs. When the utilization is 85% or lower, the LPF based approach has no clear advantage over the RTI based approach. This is because the initial guess for the RTI based approach is quite large as compared to the LPF based approach. However, the lower graph of LPF as compared to RTA and RTI is that some tasks sets are infeasible when utilization is 85% or above, and hence LPF skips testing feasibility of all tasks in those sets. This situation results in reducing the average values in the loop. Similarly, the infeasible tasks arrive early when the underlying system utilization is higher and hence all of the techniques conclude rapidly. The advantage of the LPF based scheme is clear when the underlying system utilization is kept higher. The analysis support our theoretical foundation. Therefore, we conclude that the LPF based approach is the most efficient one at higher system utilizations as compared to the rest of the existing state of the art techniques.

6.3. Time analysis of RTT techniques

In this section, we record the actual run times of the feasibility tests, that we obtained on machines at MIT CSAIL (donated by Intel) with 12-core Intel Xeon X5650 2.67 GHz CPU (12 MB L3-cache)

Please cite this article in press as: N. Min-Allah, et al., Lowest priority first based feasibility analysis of real-time systems, J. Parallel Distrib. Comput. (2013), http://dx.doi.org/10.1016/j.jpdc.2013.03.016



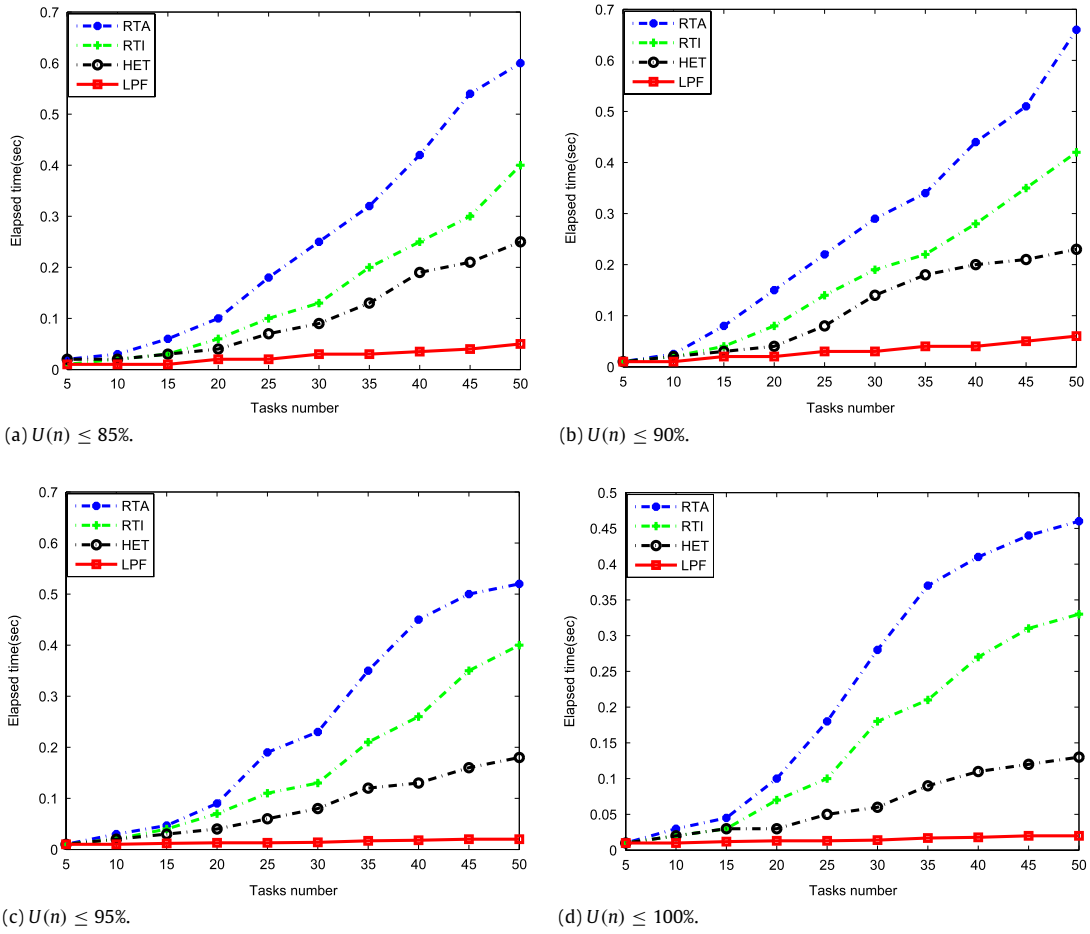
(a) $U(n) \leq 85\%$. (b) $U(n) \leq 90\%$.

(c) $U(n) \leq 95\%$. (d) $U(n) \leq 100\%$.

Fig. 3. Efficiency of LPF test over others in terms of reducing innermost loops.

and 48 GB memory. We compare our work with the RTA, RTI and HET based approaches. The results reported in Fig. 4 are plotted after 2000 runs for the tasks sets of size 5 to 50 with an increment of 5. For performance evaluation, we use the same creation as described in the beginning of Section 6. The HET tests begins with the predetermined set of scheduling points ($H_i \subseteq S_i$) and only the actual corresponding run time is extracted in this experiment. For experimental purposes, to keep the analysis aligned with the discussion made in Sections 6.1 and 6.2, we keep the system utilization in a range that gives us a clear picture under the worst case scenarios and best case scenarios for our work. By the worst case, we mean a feasible task set while the best case reflects the situations when the task set is RM infeasible. Observing Fig. 4, initially, we keep the system utilization as low as 85% so that all of the feasibility tests encounter maximum workload by analyzing the feasibility of all tasks in the set. Under varied system utilization, all of the tests exhibited similarly behavior when the task set size is smaller while the trend became more apparent when the task set is increased in size. In all given scenarios, the effectiveness of LPF is visible. It can be observed from Fig. 4(a)–(b) that the line associated with LPF technique is on the higher side as compared to Fig. 4(c)–(d). This trends is due to the fact that at lower system utilization (Fig. 4(a)–(b)) an individual task τ_i adds a lighter computation demand c_i and subsequently, the initial guess value is smaller as compared to the initial guess that would be made at higher system utilization (Fig. 4(c)–(d)). Due to this lower initial guess value, the LPF test has to undergo more iterations before determining the task schedulability. The same is true for the RTA, RTI and HET based approaches. Therefore, all of these test are faced with the maximum workload when $U(n) \leq 85\%$. When the utilization is further increased then some of the task sets become infeasible because of the higher CPU demands of individual tasks. This is due to the fact that the maximum number of tasks are schedulable under such utilization and a few tasks that belongs to the infeasible task sets portion are low priority tasks. As reflected in Fig. 4, LPF delivers better results in comparison to the other counterparts from analysis time perspective. The reason for this advantage of LPF is that the lower priority tasks are more susceptible to deadline miss than higher priority tasks and LPF starts checking feasibility with lower priority task. In such situations LPF identifies RM unschedulable task very early and skip the remaining tasks. In contrast, RTA, RTI and HET need to check the schedulability of a large number of tasks before they identify an unschedulable task. Similarly, it can be observed from Fig. 4(c) and (d) that all the tests take less time at higher utilization. This behavior is because of the existence of more unschedulable tasks in the task set generated at respective utilizations, which is understood from the existing literature [19]. The RTI based approach takes the advantage of higher initial guess over the RTA and LPF based techniques, as the computation demands of all of the tasks become higher. Therefore, the cumulative effect results in higher initial values. Due to the higher initial guess, RTI dominates RTA HET and LFP. However, the advantage that LPF exhibits over HET, RTA and RTI is its ability to determine an unschedulable task much earlier than other counterparts due to its implicit nature of investigating the RM feasibility problem from the infeasibility perspective. This is in agreement with the literature [19] that the overall system remains infeasible at higher utilization (Table 1) and hence such utilization favors LPF in determining the system infeasibility after testing only a small number (if any) of tasks. For the

Please cite this article in press as: N. Min-Allah, et al., Lowest priority first based feasibility analysis of real-time systems, J. Parallel Distrib. Comput. (2013), http://dx.doi.org/10.1016/j.jpdc.2013.03.016



(a) $U(n) \leq 85\%$. (b) $U(n) \leq 90\%$.

(c) $U(n) \leq 95\%$. (d) $U(n) \leq 100\%$.

Fig. 4. Performance dominance of LPF over RTA and RTI on various system utilization.

same infeasible task sets HET, RTA and RTI have to evaluate a considerable number of tasks before they encounter an unschedulable task and hence LPF dominates them at higher system utilization.

6.4. Worst and best cases scenarios for the proposed approach

The LPF approach turns out to be a very powerful but simple technique for analyzing the infeasibility of a real-time systems under the RM scheduling algorithm. In Section 6, we presented the results at four utilization levels that shows the supremacy LPF alternative over existing solutions. In addition to the experimentation made above, we discuss the worst and best case scenarios of the LPF in the following.

6.4.1. Worst case

The worst case scenario for our proposed technique is when Γ is feasible. Two sub cases may arise, either: (a) the system utilization is low or (b) all of the tasks are RM schedulable even when the system utilization is high. For (a), there is no need to perform the exact analysis, as the feasibility of Γ is answerable with the inexact tests that have time complexity of $O(n)$. Even then, our system can skip some tasks as it works in the LPF fashion and if τ_n is schedulable and $\digamma = \phi$, then it is very likely that feasibility of τ_{n-1} or any higher priority task τ_i is true at some instant $x \in \digamma$. With this approach, testing feasibility of remaining higher priority tasks is skipped by LPF.

When (b) is encountered, our proposed procedure determines the feasibility of n tasks in the ascending order. The same type of procedure is performed with the HPF approach. However, in the descending priority order. Although the aforementioned situation (sub case (b)) is very unlikely to occur (please refer to Table 1), the HPF based approach has no advantage over our proposed strategy in worst cases (a and b).

6.4.2. Best case

The best case scenario for our proposed technique is when the system is infeasible and possesses a higher utilization. This case is very likely to happen in reality at higher system utilizations as reported in Table 1. An infeasible task set means that at least one task τ_i is infeasible. Recall from Fig. 1 that an infeasible task is generally the lowest priority task that makes the whole task set infeasible. Therefore, determining the infeasibility of Γ with an LPF based technique means knowing the infeasibility of only a single task τ_n and skipping the remaining $(n-1)$ tasks. This situation becomes more apparent when LPF is tested under higher utilization. The experimental analysis shows the dominance of LPF approach over existing counterparts significantly when system utilization is kept high.

7. Conclusions

In this work, we explored a new dimension for the feasibility analysis of hard real-time systems under fixed priority scheduling. The proposed method was based on the observation that the lower priority tasks were more susceptible to deadline violations. The lowest priority first approach was tailored for scheduling points as well as the response time based tests. The experimental results showed that our proposed technique outperforms existing alternatives in terms of three criteria: the number of scheduling points

Please cite this article in press as: N. Min-Allah, et al., Lowest priority first based feasibility analysis of real-time systems, J. Parallel Distrib. Comput. (2013), http://dx.doi.org/10.1016/j.jpdc.2013.03.016

tested, the calls to the inner-most loops, and the actual analysis time. It was observed that under higher system utilization when the intuition is that the task set has at least one unschedulable task, LPF offers better results as compared to tests based on SPT and RTT solutions. The performance of LPF becomes more dominant when applied to a real-time system with higher system utilization.

Acknowledgments

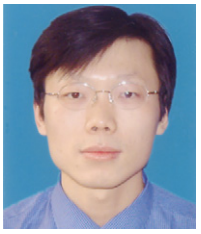
Nasro Min-Allah would like to thank the Supertech Research Group at the MIT Computer Science and Artificial Intelligence Laboratory for providing facilities to conduct experiments, when he was in residence as a Visiting Scientist.

References

[1] M. Asplund, Real Time Systems, Department of Computer and Information Science, Linköping University, Sweden, 2011, http://www.ida.liu.se/~TDDB47/lectures/tddb47-scheduling2-2x3.pdf.
[2] N.C. Audsley, On priority assignment in fixed priority Scheduling, Information Processing Letters 79 (1) (2011) 39–44.
[3] N.C. Audsley, A. Burns, M.F. Richardson, A.J. Wellings, Hard real-time scheduling: the deadline monotonic approach, in: Proceedings of 8th IEEE Workshop on Real-Time Operating Systems and Software, 1991, pp. 133–137.
[4] N.C. Audsley, A. Burns, K. Tindell, A. Wellings, Applying new scheduling theory to static priority preemptive scheduling, Software Engineering Journal (1993).
[5] E. Bini, G.C. Buttazzo, The space of rate monotonic schedulability, in: Proceedings of the 23th IEEE Real-Time Systems Symposium, Austin, Texas, 2002, pp. 169–177.
[6] E. Bini, G.C. Buttazzo, Schedulability analysis of periodic fixed priority systems, IEEE Transactions on Computers 53 (11) (2004) 1462–1473.
[7] E. Bini, G.C. Buttazzo, G. Buttazzo, A hyperplanes bound for the rate monotonic algorithm, in: Proceedings of the 13th Euromicro Conference on Real-Time Systems, 2001, pp. 59–67.
[8] A. Buchard, J. Liebeherr, Y. Oh, S.H. Son, New strategies for assigning realtime tasks to multiprocessor systems, IEEE Transactions on Computers 4 (1995) 1429–1442.
[9] A. Burns, A.J. Wellings, Real-Time Systems and Programming Languages, fourth ed., Addison Wesley, 2009, p. 602.
[10] T.C.E. Chenga, Xiuli Wang, Machine scheduling with job class setup and delivery considerations, Computers and Operations Research 37 (6) (2010) 1123–1128.
[11] M. Chrobak, L. Epstein, J. Noga, C.J. Sgall, R.V. Stee, T. Tichý, N. Vakhania, Preemptive scheduling in overloaded systems, Journal of Computer and System Sciences 67 (1) (2003) 183–197.
[12] R.I. Davis, A. Zabos, A. Burns, Efficient exact schedulability tests for fixed priority real-time systems, IEEE Transactions on Computers 57 (9) (2008) 1261–1276.
[13] N. Fisher, S. Baruah, A polynomial-time approximation scheme for feasibility analysis in static-priority systems with arbitrary relative deadlines, in: Proceedings of the 17th Euromicro Conference on Real-Time Systems, Palma de Mallorca, 2005.
[14] L. George, N. Riverre, M. Spuri, Preemptive and non-preemptive real-time uniprocessor scheduling, Research Report 2966, INRIA, France, 1996.
[15] C.C. Han, H.Y. Tyan, A better polynomial-time schedulability test for real-time static-priority scheduling algorithm, in: Proceedings of the 18th IEEE Real-Time Systems Symposium, 1997, pp. 36–45.
[16] M. Joseph, P. Pandya, Finding response times in a real-time system, The Computer Journal 29 (5) (1986) 390–395.
[17] C.M. Krishna, K.G. Shin, Real-Time Systems, McGrawHill, 1997.
[18] P.A. Laplante, et al., Real-Time Systems Design and Analysis, third ed., Wiley, IEEE Press, ISBN: 978-0-471-22855-4, 2004.
[19] P.A. Laplante, S.V. Kartalopoulos, M. Akay, M.E. El-Hawary, F.M.B. Periera, J.B. Anderson, R. Leonardi, C. Singh, R.J. Baker, M. Montrose, S. Tewksbury, J.E. Brewer, M.S. Newman, G. Zobrist, Real-Time Systems Design and Analysis, third ed., A John Wiley and Sons, 2004.
[20] J.P. Lehoczky, L. Sha, Y. Ding, The rate monotonic scheduling algorithm: exact characterization and average case behavior, in: Proceedings of the IEEE Real-Time System Symposium, 1989, pp. 166–171.
[21] J.Y.T. Leung, J. Whitehead, On the complexity of fixed-priority scheduling of periodic real-time tasks, Performance Evaluation 2 (1982) 237–250.
[22] J.W.S. Liu, Real Time Systems, Prentice Hall, 2000.
[23] C.L. Liu, J.W. Layland, Scheduling algorithms for multiprogramming in a hard real-time environment, Journal of the ACM 20 (1) (1973) 40–61.
[24] Wan-Chen Lu, Kwei-Jay Lin, Hsin-Wen Wei, Wei-Kuan Shih, Efficient exact test for rate-monotonic schedulability using large period-dependent initial values, IEEE Transactions on Computers 57 (5) (2008).
[25] Y. Manabe, S. Aoyagi, A feasibility decision algorithm for rate monotonic and deadline monotonic scheduling, Real-Time Systems 14 (2) (1998) 171–181.
[26] N. Min-Allah, I. Ali, J. Xing, Y. Wang, Utilization bound for periodic task set with composite deadline, Journal of Computers and Electrical Engineering 36 (6) (2010) 1101–1109.
[27] N. Min-Allah, H. Hussain, S.U. Khan, A.Y. Zomaya, Power efficient rate monotonic scheduling for multi-core systems, Journal of Parallel and Distributed Computing 72 (1) (2012) 48–57.
[28] N. Min-Allah, S.U. Khan, Y. Wang, Optimal task execution times for periodic tasks using nonlinear constrained optimization, The Journal of Supercomputing 59 (3) (2012) 1120–1138.
[29] N. Min-Allah, W. Yong-Ji, X. Jian-Sheng, J. Liu, Revisiting fixed priority techniques, in: EUC, Vol. 4808, in: LNCS, 2007, pp. 134–145.
[30] M. Sjödin, H. Hansson, Improved response-time analysis calculations, in: Proceedings of the 19th IEEE Real-Time Systems Symposium, 1998, pp. 399–409.
[31] K. Tei-Wei, K.M. Aloysius, Load adjustment in adaptive real-time systems, in: Proceedings of the IEEE Real-Time Systems Symposium, 1991, pp. 160–171.
[32] K.W. Tindell, Using offset information to analyze static priority pre-emptively scheduled task sets, Technical Report YCS 182, Department of Computer Science, University of York, 1992.
[33] K.W. Tindell, A. Bums, A.J. Wellings, An extendible approach for analyzing fixed priority hard real-time tasks, Real-Time Systems Journal 6 (1994) 133–151.
[34] A. Zerzelidis, A framework for flexible scheduling in real-time middleware, Ph.D. Thesis, The University of York, Department of Computer Science, 2007.

Nasro Min-Allah very successfully wears many hats: he is associated with CSAIL at Massachusetts Institute of Technology as a Visiting Scientist; he is Associate Professor and Head of the Computer Science Department at COMSATS Institute of Information Technology; he is the Director at Green Computing and Communication Lab. He received his Undergraduate and Master degrees in Electronics and Information Technology in 1998 and 2001 respectively from Quaid-i-Azam University and Hamdaramd University, Pakistan. He obtained a Ph.D. in Real-time & Embedded Systems from the Graduate University of the Chinese Academy of Sciences (GUCAS), PR China in 2008. He has been enjoying a distinguished carrier both in research and administration. He is the author of 36 research articles and book chapters published in ranking conferences and journal. His research interests include software pipelining, chromatic scheduling, reliable computing and real-time systems. He is also the winner of following three most prestigious awards: (i)-CIIT Golden Medallion for Innovation (CIMI-2009), (ii)- Best Mobile Innovation in Pakistan (BMIP-2010), and (iii) Best University Teacher Award, Pakistan (2012).

Samee U. Khan received a B.S. degree from Ghulam Ishaq Khan Institute of Engineering Sciences and Technology, Topi, Pakistan, in May 1999, and a Ph.D. from the University of Texas, Arlington, TX, USA, in August 2007. Currently, he is Assistant Professor of Electrical and Computer Engineering at the North Dakota State University, Fargo, ND, USA. Prof. Khan's research expertise encompasses topics, such as (a) Sustainable Computing: High and low level data, task, and communications schedulers, workflow managers, and application-level dynamic fine-tuning. (b) Social Networking: Disaster management, search and rescue, classroom instructional use, and malicious behavior detection. (c) Robust Backbone Networks for Cyberinfrastructures: Resource provisioning, system recovery from catastrophic anomalies, and inter-layer/inter-domain protocols. (d) Reliability: Robustness, trust, and security. In the aforementioned areas, he has published over 200 papers. He is a Fellow of the British Computer Society.



Xiuli Wang is an Associate Professor in the Central University of Finance and Economics, member of Computer Security Technical Committee of China Computer Federation. He received a Ph.D. Degree from the Institute of Software, Chinese Academy of Science in 2007. His research interests cover the following areas: information security, computer networks, data mining, optimization theory and its application.

Albert Y. Zomaya is currently the Chair Professor of High Performance Computing and Networking and Australian Research Council Professorial Fellow in the School of Information Technologies, The University of Sydney. He is also the Director of the Centre for Distributed and High Performance Computing which was established in late 2009. Professor Zomaya received his Ph.D. from the Department of Automatic Control and Systems Engineering, Sheffield University in the United Kingdom. He held visiting positions in the Department of Computer Science, Waterloo University and the Department of Computer Science, University of Missouri-Rolla. He also is an Adjunct Professor in the Department of Electrical and Electronic Engineering, at the University of Western Australia. Professor Zomaya has to his credit 19 book titles and more than 350 publications in technical journals, collaborative books, and conferences. For more information, please visit: http://sydney.edu.au/it/~zomaya/.

Please cite this article in press as: N. Min-Allah, et al., Lowest priority first based feasibility analysis of real-time systems, J. Parallel Distrib. Comput. (2013), http://dx.doi.org/10.1016/j.jpdc.2013.03.016