J. Parallel Distrib. Comput. 73 (2013) 1066–1075

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 17 April 2013

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:

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

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