Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Demand-based schedulability analysis for real-time multi-core scheduling

Jinkyu Lee[a], Insik Shin[b,*]

[a] Department of Computer Science and Engineering, Sungkyunkwan University, South Korea
[b] Department of Computer Science, KAIST, Daejeon, South Korea

ARTICLE INFO

Article history:
Received 20 January 2013
Received in revised form 30 July 2013
Accepted 19 September 2013
Available online 24 October 2013

Keywords:
Demand-based schedulability analysis
Real-time systems
Multi-core systems

ABSTRACT

In real-time systems, schedulability analysis has been widely studied to provide offline guarantees on temporal correctness, producing many analysis methods. The demand-based schedulability analysis method has a great potential for high schedulability performance and broad applicability. However, such a potential is not yet fully realized for real-time multi-core scheduling mainly due to (i) the difficulty of calculating the resource demand under dynamic priority scheduling algorithms that are favorable to multi-cores, and (ii) the lack of understanding how to combine the analysis framework with deadline-miss conditions specialized for those scheduling algorithms. Addressing those two issues, to the best of our knowledge, this paper presents the first demand-based schedulability analysis for dynamic job-priority scheduling algorithms: EDZL (Earliest Deadline first until Zero-Laxity) and LLF (Least Laxity First), which are known to be effective for real-time multi-core scheduling. To this end, we first derive demand bound functions that compute the maximum possible amount of resource demand of jobs of each task while the priority of each job can change dynamically under EDZL and LLF. Then, we develop demand-based schedulability analyses for EDZL and LLF, by incorporating those new demand bound functions into the existing demand-based analysis framework. Finally, we combine the framework with additional deadline-miss conditions specialized for those two laxity-based dynamic job-priority scheduling algorithms, yielding tighter schedulability analyses. Via simulations, we demonstrate that the proposed schedulability analyses outperform the existing schedulability analyses for EDZL and LLF.

1. Introduction

In the area of real-time systems in which temporal correctness is critical, real-time scheduling has been substantially studied to support real-time tasks without violating their own timing constraints. Those studies generally focus on two important issues: scheduling algorithms that determine the order of execution of a set of tasks, and schedulability analysis that verifies the temporal correctness of the task set under a specific scheduling algorithm. For uniprocessor systems with a sporadic task model (Mok, 1983), it has been proved that EDF (Earliest Deadline First) (Liu and Layland, 1973) is an optimal scheduling algorithm, and its demand-based schedulability analysis (Baruah et al., 1990) captures the optimality of EDF, i.e., the analysis is sufficient and necessary.

As multi-core architectures become popular due to its high computing capability with low power consumption, real-time multi-core scheduling has been paid an increasing amount of attention; many scheduling algorithms and their schedulability analyses have been adapted from real-time uniprocessor scheduling theories and/or newly developed. However, schedulability analysis methods still have much room for improvement in that most existing multi-core schedulability analyses are only sufficient, not necessary.

For tighter schedulability analysis on multi-cores, we focus on the demand-based schedulability analysis method (Baruah, 2007). Different from other existing analysis methods, such as utilization-based (Goossens et al., 2003), deadline-based (Bertogna et al., 2009) and response-time-based schedulability analysis methods (Bertogna and Cirinei, 2007), the demand-based method investigates a large number of intervals with different length, and calculates the resource requirements that a task set collectively *demands* in each interval to satisfy their individual timing constraints. Then, a task set is deemed schedulable if for all the intervals, resources can be supplied to the task set more than or equal to its resource *demand*. Hence, the demand-based method has a great potential for broad applicability and high schedulability performance. That is, once the resource demand of a task set is computed under a specific scheduling algorithm, we can easily apply the demand-based schedulability analysis framework to the algorithm. Also, the method can reduce the pessimism of

* Corresponding author. Tel.: +82 42 350 3524.
E-mail addresses: jinkyu.yi@gmail.com (J. Lee), insik.shin@cs.kaist.ac.kr (I. Shin).

0164-1212/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2013.09.029

calculating the effect of *carry-in* jobs[1] of a task set in a given interval, by limiting the contribution of carry-in jobs through the comprehensive interval check. As a result of such a tight calculation, the demand-based method yields the exact schedulability analysis of EDF on a uniprocessor platform (Baruah et al., 1990) and one of the best EDF schedulability analyses on a multi-core platform (Baruah, 2007) (See a survey Bertogna and Baruah, 2011).

While the demand-based schedulability analysis method has achieved a great success for some simple scheduling algorithms (e.g., EDF), its potential has not been fully realized for more dynamic scheduling algorithms that are suitable for multi-cores, mainly because of two reasons. First, it is challenging to calculate the amount of resource demand under those algorithms. Second, it has not been tried to combine the demand-based schedulability analysis method with additional deadline-miss conditions specialized for those algorithms, which potentially improves schedulability.

Addressing the two issues, this paper develops demand-based schedulability analyses for two dynamic job-priority scheduling algorithms: EDZL (Earliest Deadline first until Zero Laxity) (Lee, 1994) and LLF (Least Laxity First) (Leung, 1989). EDZL is a modification of EDF; it gives the highest priority to zero-laxity jobs, and schedules other jobs according EDF, where a laxity of a job at any time instant is defined as the remaining time to its deadline minus the remaining execution time at the instant. LLF prioritizes jobs according to their laxity values; the lower laxity, the higher priority. EDZL and LLF are not only proven as optimal on a uniprocessor platform (Dertouzos and Mok, 1989; Park et al., 2005) as well as EDF, but also known to be effective on a multiprocessor platform (Cho et al., 2002; Lee et al., 2010). This is because the algorithms promote zero-laxity jobs, which should be scheduled immediately to avoid deadline misses.

To develop demand-based schedulability analyses for EDZL and LLF, we need to calculate the resource demand under the algorithms. Since the priority of jobs dynamically changes under EDZL and LLF, we investigate how long a job of a later deadline can interfere another job of an earlier deadline under EDZL and LLF. Based on this, we derive demand bound functions under EDZL and LLF, and then we develop two types of schedulability analyses using the functions. First, we simply apply the functions to the existing demand-based schedulability analysis framework. Second, we incorporate the framework into additional deadline-miss conditions specialized for the laxity-based dynamic change of job priority, resulting in tighter schedulability analyses.

We demonstrate through simulation that the proposed demand-based schedulability analyses not only outperform existing schedulability analyses for EDZL (Baker et al., 2008; Lee and Shin, 2013) and LLF (Lee et al., 2010, 2012), but also find a large number of additional schedulable task sets, which are not covered by the existing schedulability analyses. As a bonus, if we apply our analyses to uniprocessors, they become exact schedulability analyses of EDZL and LLF, providing an alternative way to prove the optimality of EDZL and LLF on a uniprocessor platform.

In summary, this paper makes the following contributions.

- It calculates demand bound functions for EDZL and LLF, under which the priority of jobs dynamically changes. To the best knowledge of the authors, this study presents the first demand bound functions for dynamic job-priority scheduling algorithms;
- It develops demand-based schedulability analyses for EDZL and LLF by incorporating the functions and the existing demand-based schedulability analysis framework into additional deadline-miss conditions specialized for laxity-based dynamic priority change, thus broadening applicability of the demand-based schedulability analysis method; and
- It demonstrates the effectiveness of the proposed demand-based schedulability analyses through simulation, showing that they improve the state-of-the-art analysis methods.

The remainder of this paper is organized as follows. Section 2 introduces our system model, assumptions and notations, and then summarizes the existing demand-based schedulability analysis for EDF. Section 3 derives demand bound functions under EDZL and LLF. Section 4 develops demand-based schedulability analyses for EDZL and LLF using the derived functions. Section 5 evaluates the schedulability performance of the analyses. Section 6 discusses related work, and finally Section 7 concludes this paper.

2. Background

In this section, we first present our system model, assumptions and notations. Then, we recapitulate the existing demand-based schedulability analysis for EDF (Baruah, 2007), which will be a basis for our demand-based schedulability analyses for EDZL and LLF.

2.1. System model, assumptions and notations

In this paper we assume a sporadic task model (Mok, 1983), in which a task $\tau_i \in \tau$ is specified by (T_i, C_i, D_i), where T_i is the minimum separation, C_i is the worst-case execution time, and D_i is the relative deadline. Further, we restrict our attention to implicit $(C_i \le D_i = T_i)$ and constrained $(C_i \le D_i \le T_i)$ deadline tasks. Let $|\tau|$ denote the number of tasks in a task set τ. A task τ_i invokes a series of jobs, each separated from its predecessor by at least T_i amount of time, and supposed to finish its execution within D_i amount of time. We assume that a single job of a task cannot be executed in parallel.

We focus on a multi-core system, in which there are m identical cores. Also, we deal with global, preemptive scheduling algorithms under which jobs can be executed in any core (global), and a higher-priority job can preempt another low-priority executing job any time (preemptive). Without loss of generality, let one time unit denote the quantum length, and therefore all task parameters are specified as multiples of the quantum length.

To express a job in an interval, we use the following terms: in an interval $[t_a, t_b]$, a job is called

- *carry-in*, if the job is released before t_a and has remaining execution at t_a;
- *carry-out*, if the job is released before t_b and has remaining execution at t_b; and
- *body*, if the release time and deadline of the job are within the interval $[t_a, t_b]$.

Also, we express that a job J_1 *interferes* another job J_2 in a time slot, if J_1 executes but J_2, although ready to execute, does not execute in the time slot.

2.2. Existing demand-based schedulability analysis for EDF

The notion of resource demand has been successfully employed in many schedulability analyses on various platforms (Baruah et al., 1990; Shin and Lee, 2003; Baruah, 2007). The demand of a given interval represents the amount of execution that should be performed in order to finish the body jobs without any deadline miss. A task set is schedulable under a scheduling algorithm if the demand of each interval is no greater than the amount of resource supplied within the same interval. A key issue is then how to calculate the demand of any given interval accurately, and it requires a good

[1] A job is said to be a carry-in job in an interval, if it is released before the interval and has remaining execution at the beginning of the interval.