Open Problems in Scheduling Self-Suspending Tasks

Karthik Lakshmanan[†], Shinpei Kato[†‡], and Ragunathan (Raj) Rajkumar[†]
[†]Department of Electrical and Computer Engineering, Carnegie Mellon University
[‡]Department of Computer Science, The University of Tokyo

Abstract

Self-suspension intervals are becoming increasingly common in various systems such as: (i) multi-core processors, where tasks running on one core have to synchronize with tasks running on other cores, (ii) heterogeneous ISA multi-core processors, where certain instructions can only be executed on specific processor cores, and (iii) systems with accelerated co-processors such as Digital Signal Processors (DSPs) or Graphics Processing Units (GPUs). In the light of these developments, a few key questions arise: (a) What should be the standard task model for specifying suspension intervals in such systems? (b) Given that classical scheduling algorithms such as Earliest-Deadline First (EDF) suffer from scheduling anomalies in self-suspending task systems, does there exist a competitive anomaly-free scheduling algorithm for such systems? (c) Given that the feasibility problem of scheduling periodic tasks with at most one self-suspension per task and implicit deadlines is NP-Hard, what can we say about the feasibility condition for scheduling sporadic self-suspending task systems? In this position paper, we provide some preliminary ideas and intuitions towards answering these questions, and seek to engage the broader real-time research community in solving these open problems.

1. Problem Context

Recent years have seen significant changes in the landscape of processor technologies, ranging from the emerging trend of massively multi-core processors to general-purpose computing support in Graphics Processing Units (GPUs). These technology trends project a future in which computation is no longer independently carried out on one processor core but needs to be often synchronized with other processor cores and GPUs. Synchronizing with external events results in suspension intervals, where tasks voluntarily release control of the processor for extended invervals of time. In such systems, classical real-time task models need to be augmented with upper bounds on the duration of suspension intervals. This is an important problem since dealing with suspension intervals as just part of the computation time itself is not a scalable approach as tasks on each processor core are going to increasingly rely on other processor cores and co-processors.

2. Task Models

Developing an useful and practical task model is the first basic requirement for advancing the state of the art in analyzing self-suspending tasks. Traditional approaches have looked at the following task models:

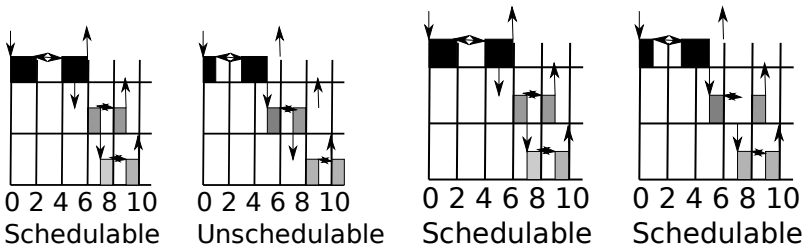
2.1 The $\tau : (C, E, T, D)$ Traditional Model

τ is a task that releases jobs with a minimum inter-arrival time of T time units, each job of τ has a relative deadline of D time units from its release, C is an upper bound on the total execution time of each job of τ, and E is an upper bound on the total suspension time for each job of τ.

From an analysis perspective, this leads to quite pessimistic results since the locations of the suspensions within the jobs of τ are unknown. From a system designer perspective, this is a task model that is easy to specify and reason about. It alleviates the burden of finding the exact number of suspension intervals and their locations with respect to job releases. This approach has been used in work reported in [5, 3, 4].

2.2 The $\tau : ((C^1, E^1, C^2, E^2, ..., C^m), T, D)$ Extended Model

τ is a task that releases jobs with a minimum inter-arrival time of T time units, each job of τ has a relative deadline of D time units from its release. The execution of each job of τ comprises of m computation segments interleaved by $m-1$ suspension intervals. This model subsumes the (C, E, T, D) model in that tasks in the $((C^1, E^1, C^2, E^2, ..., C^m), T, D)$ model can be modeled using (C, E, T, D) as $(\sum_{j=1}^{m} C^j, \sum_{k=1}^{m-1} E^k, T, D)$. This approach has been used in [2, 6].



(a) Anomaly under EDF (b) EDZL Scheduling

Figure 1. Scheduling anomaly with self-suspending tasks under EDF and prevention using EDZL

2.3 Computation Graphs

A more exhaustive approach could be to model the computation of each job as a Directed Acyclic Graph (DAG), which enables the modeling of branches and leads to a more precise representation of the task. Analyzing the schedulability of such DAGs might be more complex but it has the best potential for achieving an exact-case analysis since it subsumes both the (C, E, T, D) and $((C^1, E^1, C^2, E^2, ..., C^m), T, D)$ models.

An open question related to this is the following: *What is a practical and useful model for specifying suspension intervals in real-time tasks?*

3. Scheduling Anomalies from Self-Suspension

Reference [6] showed that scheduling anomalies can arise from using classical scheduling algorithms such as EDF in self-suspending task systems. The example task set used in [6] is given in Figure 1(a), where a job J_1 with execution pattern $(2, 2, 2)$ and deadline 6 is released at time 0, job J_2 with execution pattern $(1, 1, 1)$ and deadline 4 is released at time 5, and job J_3 with execution pattern $(1, 1, 1)$ and deadline 3 is released at time 7. Reducing the execution-time requirement of the first segment of J_1 by 1 time unit causes J_3 to miss its deadline under EDF, as shown in Figure 1(a).

An open question is *whether there exists a competitive anomaly-free scheduling algorithm for such task systems with better performance than EDF.*

In terms of answering this question, it is known that *EDZL* (Earliest-Deadline First with Zero-Laxity) [1] performs better than EDF on *non-suspending* tasks. An interesting point to start would be applying *EDZL* (Earliest-Deadline First with Zero-Laxity) [1] scheduling to *self-suspending* tasks. We show in Figure 1(b) that EDZL can avoid the anomaly introduced in EDF.

An open question related to this is *Does dynamic-priority scheduling strictly outperform fixed-priority scheduling in self-suspending task systems?*

4. Self-Suspending Sporadic Task Systems

It is known from [6] that the feasibility problem of scheduling periodic tasks with at most one self-suspension per task and implicit deadlines is NP-hard in the strong sense. However, it remains to be seen whether a polynomial time or pseudo-polynomial time exact-case feasibility condition can be developed for sporadic self-suspending task systems, due to their potential for having a more simpler characterization of the worst case.

An open question is *whether we can develop an anomaly-free exact feasibility condition for self-suspending sporadic task systems*.

In this regard, with respect to fixed-priority scheduling, [2] provides an exact characterization of the critical scheduling instant for a self-suspending task with respect to interference from higher-priority non-suspending sporadic tasks.

References

[1] T. P. Baker, M. Cirinei, and M. Bertogna. Edzl scheduling analysis. *Real-Time Syst.*, 40(3):264–289, 2008.

[2] K. Lakshmanan and R. R. Rajkumar. Scheduling self-suspending real-time tasks with rate-monotonic priorities. In *RTAS '10: Proceedings of RTAS 2010*, 2010.

[3] C. Liu and J. H. Anderson. Supporting sporadic pipelined tasks with early-releasing in soft real-time multiprocessor systems. In *RTCSA '09: Proceedings of the 2009 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 284–293, Washington, DC, USA, 2009. IEEE Computer Society.

[4] C. Liu and J. H. Anderson. Task scheduling with self-suspensions in soft real-time multiprocessor systems. In *RTSS '09: Proceedings of the 2009 30th IEEE Real-Time Systems Symposium*, pages 425–436, Washington, DC, USA, 2009. IEEE Computer Society.

[5] J. Liu. In *Real-Time Systems*. prentice hall, 2000.

[6] F. Ridouard, P. Richard, and F. Cottet. Negative results for scheduling independent hard real-time tasks with self-suspensions. In *RTSS '04: Proceedings of the 25th IEEE International Real-Time Systems Symposium*, pages 47–56, Washington, DC, USA, 2004. IEEE Computer Society.