Reliability-Aware Energy Minimization for Real-Time Embedded Systems with Window-Constraints

Linwei Niu and Luis Medina
Department of Computer and Electrical Engineering and Computer Science
California State University Bakersfield
Bakersfield, CA
{lniu, lmedina}@csub.edu

Yiran Chen
Department of Electrical and Computer Engineering
University of Pittsburgh
Pittsburgh, PA
yic52@pitt.edu

ABSTRACT

In this work, we propose a reliability-aware energy minimization scheme that can satisfy the *window-constraints*, *i.e.*, no more than x_i deadlines are missed in each nonoverlapped sequence of y_i jobs in real-time task τ_i. The simulation reveals that our proposed techniques can outperform previous ones in energy reduction for real-time systems with reliability awareness under *window-constraints*.

1. INTRODUCTION

With system reliability in mind, plenty of works have been published in reducing the energy consumption for *hard* real-time systems. However, few real-time applications are truly hard. For example, in network packet streams, some deadline misses are allowed provided that user's perceived quality of service (QoS) constraints are satisfied. The *window-constrained* model [2] is a very suitable QoS model for such kind of applications [2, 1]. In [1] a technique was proposed that could ensure the *window-constraints* based on *evenly distributed pattern* such as in Figure 1(a). On the other hand, to preserve the system reliability, at the end of each mandatory job, a portion of available slack could be reserved for scheduling a recovery job for it [3]. If the mandatory job failed, the recovery job will be invoked and executed [3].

Note that in the above approach, in order to reserve space for the recovery jobs, the system feasibility could be affected. For example, for the task set in Figure 1(a), after we reserved a recovery job for each mandatory job before its deadline (for example, R_{10} for J_{10}), the system became overloaded and deadline missing would become inevitable. Fortunately, under *window-constrained* model, the recover jobs could be scheduled more flexibly. For example, for the task set in Figure 1(a), if we utilize the space belonging to the optional jobs to help schedule the recovery jobs, as shown in Figure 1(b), the task set could be well schedulable. Moreover, considering the large run-time variations in real-time systems, it would be extremely profitable to adjust the patterns as well as scale the job speeds dynamically. The idea is, if some optional jobs with short actual execution times could meet their deadlines with relatively lower speeds, the corresponding future mandatory (and recovery) jobs could be demoted to optional or even be dropped without execution, as shown in Figure 1(c) and (d). It is not hard to see that in Figure 1(d), since some optional jobs (for example, J_{20} and J_{30}) could meet deadlines with much lower speeds (s_{20} and s_{30}), significant energy could be saved compared to the schedule in Figure 1(b).

To formalize the above procedure, in our work, we first derived heuristics to determine the static job/recovery patterns. Then based on it we developed dynamic pattern variation algorithms to accommodate the dynamic nature of real-time systems.

2. EVALUATION AND CONCLUSIONS

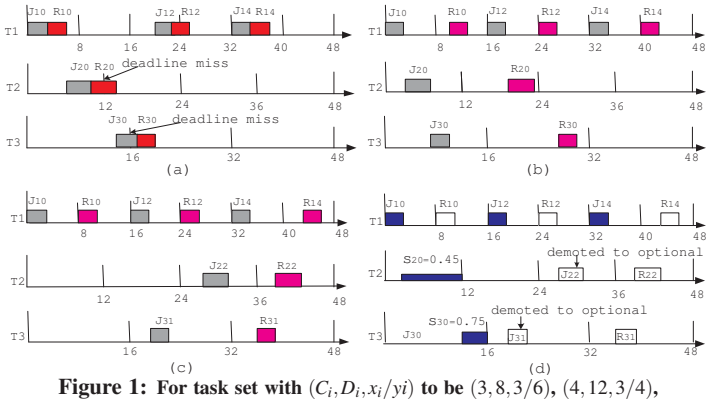

Figure 1: For task set with $(C_i, D_i, x_i/yi)$ to be $(3,8,3/6)$, $(4,12,3/4)$, and $(3,16,2/3)$, respectively: (a) under evenly distributed patterns; (b) using optional jobs to fulfill recovery jobs; (c) varied static job/recovery patterns; (d) dynamic job/recovery patterns with speed scaling.

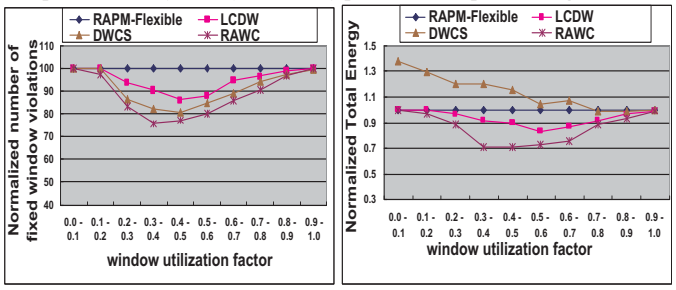

Figure 2: Comparison on: (a) window violations; (b) energy.

Our experiments adopted the same task and energy model as in [1] and fault/recovery models as in [3]. As shown in Figure 2(a), in most utilization intervals, Algorithm **DWCS** [2] has less window violations than **RAPM-Flexible** [3], and **LCDW** [1]. However, its energy consumption is also the highest (Figure 2(b)) because it executed a lot of redundant jobs. In all cases, our algorithm, *i.e.*, **RAWC** has the least number of window violations. Moreover, from Figure 2(b), **RAWC** can achieve significant energy reduction compared with all the other approaches.

And we proved that our dynamic algorithm can satisfy the window-constraints if task sets are schedulable under static recovery pattern.

3. REFERENCES

[1] N. Linwei. Energy efficient scheduling for real-time systems with qos guarantee. *Journal of Real-Time Systems*, 47(2):75–108, 2011.

[2] R. West, Y. Zhang, K. Schwan, and C. Poellabauer. Dynamic window-constrained scheduling of real-time streams in media servers. *IEEE Trans. on Computers*, 53(6):744–759, June 2004.

[3] D. Zhu, X. Qi, and H. Aydin. Energy management for periodic real-time tasks with variable assurance requirements. In *RTCSA*, 2008.