Work-In-Progress: Enhanced Energy-Aware Standby-Sparing Techniques for Fixed-Priority Hard Real-Time Systems

Linwei Niu and Jonathan Musselwhite
Department of Math and Computer Science
West Virginia State University
{*lniu, jmus*}*@wvstateu.edu*

Wei Li
Department of Computer and Electrical Engineering and Computer Science
California State University Bakersfield
wli@csub.edu

***Abstract*—For real-time computing systems, energy efficiency and reliability are two primary design concerns. In this research work, we study the problem of enhanced energy-aware standby-sparing for fixed-priority (FP) hard real-time systems under reliability requirement. The standby-sparing system adopts a primary processor and a spare processor to provide fault tolerance for both permanent and transient faults. In order to keep the energy consumption for such kind of systems under control, we explore enhanced fixed-priority scheduling schemes to minimize the overlapped concurrent executions of the workloads on the primary processor and on the spare processor, enabling energy savings. Moreover, efficient online scheduling techniques are under development to boost the energy savings during run-time while preserving the system reliability.**

I. Introduction

Energy conservation has come to be recognized as a critical design issue for pervasive real-time embedded systems. With the aggressive scaling of transistor size in CMOS circuits, more and more transistors are integrated into a single die and the power consumption of IC chips has been increasing dramatically. Power aware scheduling has proven to be an effectively way to reduce the power consumption. For the past two decades, extensive researches on power aware scheduling techniques (e.g. [1], [2]) have been reported on energy minimization for embedded real-time systems. In the mean time, reliability has also been a major concern in the design of fault tolerant computing systems as system fault(s) could occur anytime during the execution cycle of real-time jobs [2]. Generally, computing system faults can be classified into permanent faults and transient faults [3]. Permanent faults could be caused by hardware failure or permanent damage in processing unit(s) whereas transient faults are mainly caused by temporary factors such as electromagnetic interference and cosmic ray radiations.

In recently years plenty of works (e.g. [4]) have been reported in energy conservation for fault-tolerant real-time systems. Many of them are focused on dealing with transient faults. A widely adopted strategy is to reserve a recovery job, whenever possible, for the job subject to transient fault(s). If a job is found to have failed, its recovery job, if any, will be invoked for execution to compensate the failed job.

In more recent years the *standby-sparing* technique has gained much attention in research community for its capability of tolerating both permanent and transient faults. The standby-sparing makes use of the hardware redundancy in multi-core/multiprocessor systems. Specifically, a standby-sparing system consists of two processors, a primary one and a spare one, executing in parallel. For each real-time job executed on the primary processor, there is a corresponding backup job reserved for it on the spare processor [5]. As such, whenever a permanent fault occurs to the primary or the spare processor, either of them, the other one can still continue to execute without causing system failure. Moreover, it is not hard to see that the backup jobs on the spare processor can also help tolerate transient faults (which occurs more frequently than permanent faults [6]) for their corresponding main jobs on the primary processor.

Since in a standby-sparing system, both the primary processor and the spare processor need to be executed in parallel, its total energy consumption could be quite considerable. Regarding that, some research works have been reported in literature to reduce energy consumption (e.g. [7], [8], [5]). The main idea is to try to let the workloads of the main jobs on the primary processor and their corresponding backup jobs on the spare processor be shifted away as much as possible, for example, having the main jobs on the primary processor executed as soon as possible while having the backup jobs on the spare processor executed as late as possible. Once the main jobs are completed successfully, their corresponding backup jobs could be canceled early, enabling energy saving on the spare processor. Note that the delayed execution of the jobs on the spare processor should not cause any deadline missing. In [8], Haque *et. al* employed Earliest-Deadline-First (EDF) and Earliest-Deadline-Late (EDL) schemes for jobs executed on the primary and spare processors, respectively, allowing delayed execution for the backup jobs on the spare processor, therefore saving energy. Although EDF and EDL schemes can support systems with higher utilization than fixed-priority (FP) scheme, they are difficult to implement in commercial kernels which do not provide explicit support for timing constraints, such as periods and deadlines [9]. Moreover, when the system is overloaded, EDF and EDL can produce unbounded and unpredictable deadline missings whereas FP has better stability in such cases because it is always known in advance which task will miss its deadline. Due to its high predicability, low overhead, and ease of implementation, FP scheme has been widely adopted. Mohammad [5] *et. al* proposed a standby-sparing technique for FP real-time systems based on dual priority scheme. In this approach, the workloads for all jobs

on the spare processor are procrastinated with a fixed length of time interval called "promotion time". Although this approach can help delay the backup jobs on the spare processor to some extent, its energy efficiency could be limited by the "promotion time" calculated based on the worst case response time. In this work, we will explore the enhanced standby-sparing techniques under fixed priority assignment to achieve better energy efficiency.

II. Preliminaries

A. System models

The hard real-time system considered in this paper contains n independent periodic tasks, $\mathcal{T} = \{\tau_1, \tau_1, \cdots, \tau_n\}$, scheduled according to the fixed-priority (FP) scheduling algorithm. Each task contains an infinite sequence of periodically arriving instances called *jobs*. Task τ_i is characterized with a three-tuple (P_i, D_i, C_i), denoting its period, relative deadline, and the worst case execution time (WCET), respectively.

Each task τ_i has a corresponding backup task $\tau_i^{'}$ with the same timing parameters as τ_i.

The system architecture consists of a dual processor standby-sparing system, with one be the primary and the other one be the spare [7]. Different from most of the existing works in which all tasks in $\mathcal{T}$ are executed on the primary processor and all backup tasks are executed on the spare processor, in this work, any task τ_i in $\mathcal{T}$ and its backup task $\tau_i^{'}$, either one of them, can be executed on the primary processor or the spare one so long as they are not on the same processor. For convenience, we call the original copy of the task(s)/job(s) *main task(s)/job(s)*.

Both the primary processor and the spare processor can be put into low power state when there is no pending job execution and be waken up later when idle time expired. We denote the primary/spare processor power with P_{act} when running a task, and P_{idle} when it is idle (yet still *on*). When the processor is sleeping, its power consumption is denoted as P_{sleep}.

B. Fault Model

Similar to the standby-sparing systems in [8], [5], the system we considered can tolerate both permanent and transient faults. Note that we only take the permanent fault of the processor into consideration and don't consider the permanent fault of other components in the system. With the redundancy of the processing units, our system can tolerate at most one permanent fault on the primary or on the spare processor. For transient faults, since they are more frequent than permanent faults [6] and can occur anytime during the task execution, each job is subject to transient fault(s). We assume that transient fault(s) can be detected at the end of a job's execution using sanity (or consistency) checks [10] and the overhead for detection can be integrated into the job's execution time. Transient faults are addressed through backup jobs on the other processor. Specifically, Transient faults are compensated by executing the backup jobs in the same time frame as their corresponding main jobs. Whenever the main job is completed successfully, the remaining part of its backup job will be canceled immediately. Otherwise if a fault is detected at the end of a main job's execution, its backup job will have to be executed until it is finished.

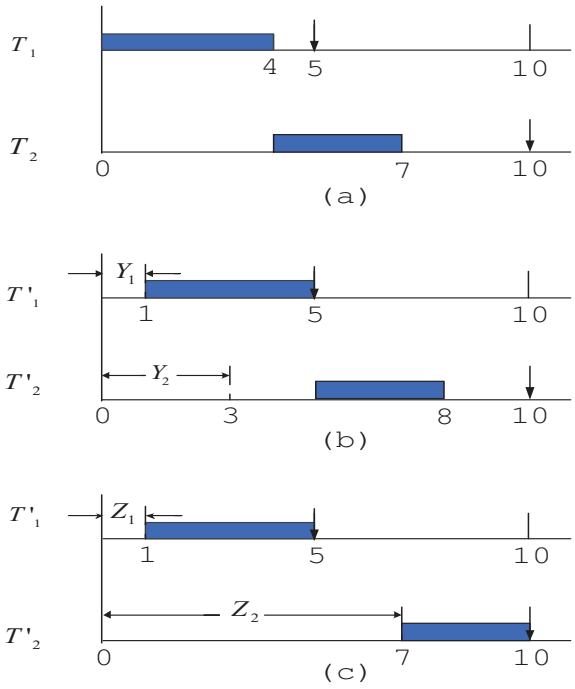

Fig. 1. (a) The schedule for the main tasks $\{\tau_1 = (10,5,4); \tau_2 = (10,10,3)\}$ on the primary processor under the deadline monotonic scheduling (DMS) scheme; (b) The schedule for the backup tasks on the spare processor with promotion time Y_1 and Y_2 calculated based on equation (1); (c) The schedule for the backup tasks on the spare processor with optimal procrastination time $Z_1 = 1$ and $Z_2 = 7$.

Similar to [2], in this work, we assumed that transient faults follow Poisson distribution and adopted the same exponential fault rate model proposed in [2]. However, the application of our work is not limited to the exponential fault rate model. One should be able to adopt any valid reliability model with appropriate reliability requirements in the approaches proposed in this paper.

III. Motivations

Our goal is to reduce energy consumption for standby-sparing systems. From the above system models, it is not hard to see that, due to the overlapped concurrent executions of the main job(s) and their corresponding backup job(s) (to provide system reliability), the most efficient way to save energy is to let the workloads of the main job(s) on one processor and the backup jobs on the other processor be shifted away as much as possible. To achieve this goal, in [5] Mohammad *et. al* proposed to run the main tasks on the primary processor according to regular FP scheme and the backup tasks on the spare processor according to **dual priority** scheme. Their approach is based on the concept of "promotion time" (denoted as Y_i), calculated as followed:

$$Y_i = D_i - R_i \tag{1}$$

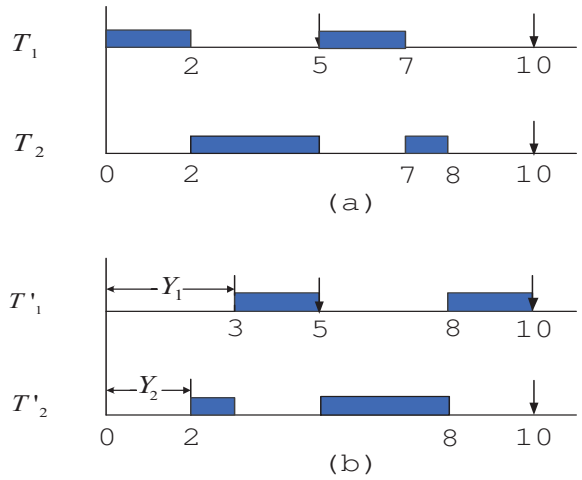

Fig. 2. (a) The schedule for the main tasks $\{\tau_1 = (5,5,2); \tau_2 = (10,10,4)\}$ on the primary processor under the deadline monotonic scheduling (DMS) scheme; (b) The backup tasks τ'_1 and τ'_2 on the spare processor with procrastination times ($Y_1 = 3$ and $Y_2 = 2$) calculated based on equation (1) cannot be delayed further.

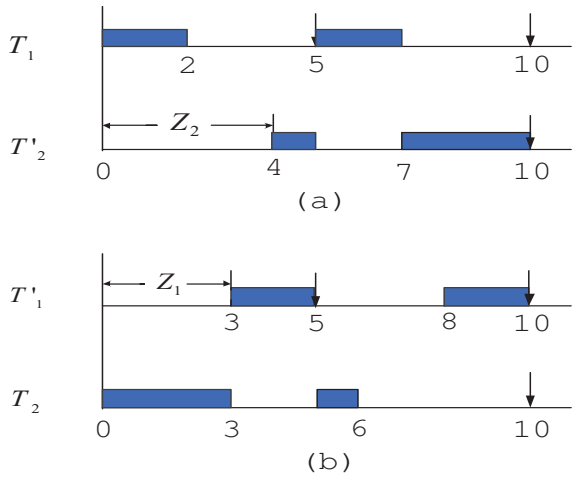

Fig. 3. (a) The schedule for the main tasks τ_1 and backup task τ'_2 on the primary processor, with procrastination time for τ'_2 extended to $Z_2 = 4$; (b) The schedule for the backup tasks on the spare processor with procrastination time for τ'_1 calculated as $Z_1 = 3$.

where R_i is the worst case response time of task τ_i.

By applying dual priority, each backup job from backup task τ'_i on the spare processor could be procrastinated by Y_i time units. Although this approach can help reduce the overlapped execution between the main job and its backup job to some extent, the procrastination time Y_i calculated above is not necessarily optimal. Actually it is only a lower bound of the allowable procrastination time for all jobs of backup task τ'_i. This could be illustrated in the following example.

Consider a task set of two tasks, *i.e.*, $\tau_1 = (10,5,4)$, $\tau_2 = (10,10,3)$, to be executed in a standby-sparing system under deadline monotonic scheduling (DMS) scheme, which is the optimal scheme for FP scheduling policy. The schedule for the main tasks on the primary processor is shown in Figure 1(a). Meanwhile, from equation (1), the promotion times Y_1 and Y_2 for tasks τ_1 and τ_2 are calculated as 1 and 3, respectively. Based on them the delayed executions of the backup tasks on the spare processor are shown in Figure 1(b). It is not hard to see that, in this case, the executions of task τ_2 in Figure 1(a) and backup task τ'_2 in Figure 1(b) have 2 time units overlapped.

However, a different schedule in Figure 1(c) with the procrastination times for backup tasks τ'_1 and τ'_2 be re-calculated as $Z_1 = 1$ and $Z_2 = 7$, respectively, could reach an overlapped execution of 0 time units between τ_2 on the primary processor (Figure 1(a)) and its backup τ'_2 on the spare processor (Figure 1(c)), therefore achieving better energy efficiency.

From the above example we can see that the procrastination of backup jobs based on the promotion time computed with equation (1) is not necessarily most efficient in terms of delaying the job execution(s) and it is possible to procrastinate the backup job(s) further to enable better energy savings.

Moreover, even when the backup job(s) cannot be delayed further, there could still be opportunity to re-organize the allocations of the main tasks and the backup tasks on the primary/spare processors to enhance energy savings. We will illustrate that with another example.

Consider a different task set of two tasks, *i.e.*, $\tau_1 = (5,5,2)$, $\tau_2 = (10,10,4)$, to be executed in a standby-sparing system. The schedule for the main tasks on the primary processor is shown in Figure 2(a). From equation (1), the promotion times for tasks τ_1 and τ_2 are calculated as $Y_1 = 3$ and $Y_2 = 2$, respectively. Based on them, as shown in Figure 2(b), Y_1 and Y_2 are already the maximal times that backup tasks τ'_1 and τ'_2 can be delayed to and it is not possible to delay them further. As a result, the executions of task τ_2 on the primary processor and τ'_2 on the spare processor will have 4 time units overlapped. Since in this case the overlapped execution time of τ_1 and τ'_1 is 0, the total overlapped executions of the tasks on the primary/spare processors are 4 time units.

However, if we re-allocate the main tasks and the backup tasks on the primary and spare processors in a different way, as in Figure 3, it is possible to reduce the overlapped execution time between the main task(s) and the backup task(s) further. As shown in Figure 3(a) and (b), this time we let τ_1 and τ'_2 be allocated on the primary processor while letting τ'_1 and τ_2 be allocated on the spare processor. With such new allocation, τ'_1 can be delayed by 3 time units (Z_1 in Figure 3(b)) while τ_2 can be delayed by 4 time units (Z_2 in Figure 3(a)). It is not hard to see that, under such new configuration, all tasks are feasible and the overlapped execution time of task τ_2 and τ'_2 is reduced to 1 time unit (while the overlapped execution time of τ_1 and τ'_1 is still 0). As a result the total overlapped execution time of the tasks on the primary/spare processors are reduced to 1 time unit, much less than that in Figure 2.

IV. Work in progress

As shown in the above example in Figure 1, the procrastination time for the backup jobs computed based on equation (1) is not necessarily optimal. Actually the promotion time Y_i based on equation (1) is a lower bound for the allowable procrastination time for all jobs of backup task τ'_i. In this research we explored new ways to find the maximal procrastination time for each backup job to enable better energy savings.

As also shown in the example in Figure 2 and Figure 3, when the main tasks and backup tasks are allocated on both

processors in a hybrid way, there are opportunities to delay the backup jobs further to boost energy savings. In this regard we are also exploring ways to enhance the calculation of the procrastination time of the backup jobs under hybrid allocation of the main/backup tasks on both processors, thus to achieve better energy efficiency. Moreover, efficient online scheduling techniques are under development to improve the energy performance of standby-sparing systems further during run-time.

V. Preliminary results

In this part, we provide some preliminary results to demonstrate the energy saving potential of our newly proposed approach for standby-sparing systems. Specifically, we compared the performance of the newly proposed approach with two other existing approaches in literature using simulation. The approaches compared are as followed:

- *SSNEM* The task sets are executed in an standby-sparing system without energy management. We use its results as the reference results;
- *SSFP* The approach in [5], which adopts dual priority scheme to delay the backup jobs in the spare processor;
- *EFPSS* our proposed enhanced fixed priority scheduling scheme for standby-sparing systems (section IV).

The periodic task set in our experiments consisted of no more than five tasks. Each task set was randomly generated with the periods randomly chosen in the range of $[10, 50]ms$. We assumed that the deadlines for the tasks were less than or equal to their periods. The worst case execution time (WCET) was set to be uniformly distributed from 1 to the deadline. To investigate energy performance of the different approaches under different workload, we divided the total utilization, *i.e.*, $\sum_i \frac{C_i}{T_i}$, into intervals of length 0.1. To reduce the statistical errors, we required that each interval contain at least 20 task sets schedulable under deadline monotonic scheme (DMS), or until at least 5000 task sets had been generated within each interval.

For the processor model we adopted a high-end dual-core processor model, *i.e.*, the Intel Core2 Duo E6850 Processor [11]. According to [11], the Intel Core2 Duo E6850 Processor can operate at a maximal speed of 3.006 GHz with power consumption 54.236 Watts .

For the fault model we adopted the same transient fault model as in [2], *i.e.*, the transient faults are assumed to follow the Poisson distribution with an average fault rate of 10^{-6} at the maximum speed. Meanwhile, we assume that at most one permanent fault could ever occur during the system running.

The energy performance of the different approaches (normalized to that by *SSNEM*) are shown in Figure 4. From Figure 4, we can see that the newly proposed approach, *i.e.*, *EFPSS* can have better energy performance than the previous approaches. This is because, by delaying each backup job based on our newly calculated procrastination time, *EFPSS* can reduce the overlapped executions between the main job(s) and their corresponding backup job(s) more effectively. Once the main job(s) are completed successfully, their backup job(s) can be canceled earlier in time, thus reducing the redundant energy cost for executing the remaining part of the backup jobs. Moreover, with the proposed hybrid main-task(s)/backup-task(s) re-allocation strategy, the energy performance of *EFPSS* could be boosted more efficiently because, in this context, the backup task(s) could be delayed more effectively, reducing more time of overlapped executions between the main tasks and the backup tasks. From Figure 4, compared with *SSFP*, the maximal energy reduction by *EFPSS* can be around 18%.

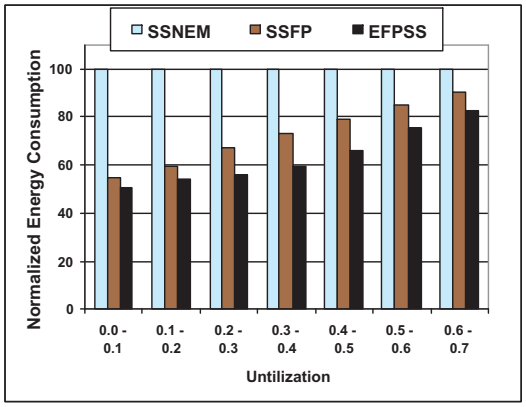

Fig. 4. Comparison of energy consumptions of the different approaches.

Acknowledge*

This work is supported in part by NSF under project HRD-1800403.

References

[1] L. Niu and D. Zhu, "Reliability-aware scheduling for reducing system-wide energy consumption for weakly hard real-time systems," *Journal of Systems Architecture*, vol. 78, pp. 30 – 54, 2017.

[2] D. Zhu, R. Melhem, and D. Mosse, "The effects of energy management on reliability in real-time embedded systems," in *Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design*, ser. ICCAD '04, 2004, pp. 35–40.

[3] B. P. R. J. J. Srinivasan, A. S.V. and C.-K. Hu, "Ramp: A model for reliability aware microprocessor design," *IBM Research Report, RC23048*, 2003.

[4] D. Zhu, "Reliability-aware dynamic energy management in dependable embedded real-time systems," *ACM Trans. Embed. Comput. Syst.*, vol. 10, pp. 26:1–26:27, January 2011.

[5] M. A. Haque, H. Aydin, and D. Zhu, "Energy-aware standby-sparing for fixed-priority real-time task sets," *Sustainable Computing: Informatics and Systems*, vol. 6, pp. 81 – 93, 2015.

[6] X. Castillo, S. R. McConnel, and D. P. Siewiorek, "Derivation and calibration of a transient error reliability model," *IEEE Trans. Comput.*, vol. 31, pp. 658–671, July 1982.

[7] A. Ejlali, B. M. Al-Hashimi, and P. Eles, "Low-energy standby-sparing for hard real-time systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 3, pp. 329–342, March 2012.

[8] M. A. Haque, H. Aydin, and D. Zhu, "Energy-aware standby-sparing technique for periodic real-time applications," in *2011 IEEE 29th International Conference on Computer Design (ICCD)*, Oct 2011, pp. 190–197.

[9] G. Buttazzo, "Rate monotonic vs. edf: Judgement day," *Real-Time Systems*, vol. 29, no. 1, pp. 5–26, 2005.

[10] D. K. Pradhan, Ed., *Fault-tolerant Computing: Theory and Techniques; Vol. 2*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1986.

[11] S. Park, J. Park, D. Shin, Y. Wang, Q. Xie, M. Pedram, and N. Chang, "Accurate modeling of the delay and energy overhead of dynamic voltage and frequency scaling in modern microprocessors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 5, pp. 695–708, May 2013.