Journal of Systems Architecture 78 (2017) 68–80

Contents lists available at ScienceDirect

Journal of Systems Architecture

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

Exploiting primary/backup mechanism for energy efficiency in dependable real-time systems

Yifeng Guo[a], Dakai Zhu[a,*], Hakan Aydin[b], Jian-Jun Han[c], Laurence T. Yang[d]

[a] Department of Computer Science, The University of Texas at San Antonio, San Antonio, TX 78249, USA
[b] Department of Computer Science, George Mason University, Fairfax, VA 22030, USA
[c] School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China
[d] Department of Computer Science, St. Francis Xavier University, Antigonish, NS, B2G 2W5, Canada

ARTICLE INFO

Article history:
Received 17 December 2016
Revised 20 March 2017
Accepted 29 June 2017
Available online 30 June 2017

Keywords:
Real-time systems
Multiprocessor
Fault tolerance
Primary/backup
Energy management
DVFS
DPM

ABSTRACT

Primary/Backup has been well studied as an effective fault-tolerance technique. In this paper, with the objectives of tolerating a single *permanent* fault and maintaining system reliability with respect to *transient* faults, we study dynamic-priority based energy-efficient fault-tolerance scheduling algorithms for periodic real-time tasks running on multiprocessor systems by exploiting the primary/backup technique while considering the negative effects of the widely deployed *Dynamic Voltage and Frequency Scaling (DVFS)* on transient faults. Specifically, by separating primary and backup tasks on their dedicated processors, we first devise two schemes based on the idea of *Standby-Sparing (SS)*: For *Paired-SS*, processors are organized as groups of two (i.e., pairs) and the existing SS scheme is applied within each pair of processors after partitioning tasks to the pairs. In *Generalized-SS*, processors are divided into two groups (of potentially different sizes), which are denoted as *primary* and *secondary* processor groups, respectively. The main (backup) tasks are scheduled on the primary (secondary) processor group under the *partitioned-EDF* (*partitioned-EDL*) with DVFS (DPM) to save energy. Moreover, we propose schemes that allocate primary and backup tasks in a *mixed* manner to better utilize system slack on all processors for more energy savings. On each processor, the *Preference-Oriented Earliest Deadline (POED)* scheduler is adopted to run primary tasks at scaled frequencies *as soon as possible (ASAP)* and backup tasks at the maximum frequency *as late as possible (ALAP)* to save energy. Our empirical evaluations show that, for systems with a given number of processors, there normally exists a configuration for Generalized-SS with different number of processors in primary and backup groups, which leads to better energy savings when compared to that of the Paired-SS scheme. Moreover, the POED-based schemes normally have more stable performance and can achieve better energy savings.

© 2017 Elsevier B.V. All rights reserved.

1. Introduction

Fault tolerance has been a traditional research topic in real-time systems as computing devices are subject to different types of faults at runtime. In general, to tolerate various faults and guarantee that real-time tasks can complete their executions successfully on time, the existing fault tolerance techniques normally adopt different forms of redundancy. For instance, as a simple and well-studied approach, hot-standby exploits hardware/modular redundancy and runs two copies of the same task concurrently on two processors to tolerate a single fault [33]. However, by their very nature, such redundancy-based fault-tolerance techniques demand more system resources, which can lead to excessive energy consumption (e.g., hot-standby has 100% energy overhead).

On the other hand, with the ever-increasing power density in modern computing systems, energy has been promoted as a first-class system resource and energy-aware computing has become an important research area [24]. As a common energy saving technique, *dynamic power management (DPM)* can power down (or turn off) components when they are not in use. Moreover, as a fine-grained power management technique, *dynamic voltage and frequency scaling (DVFS)* can operate computing systems at different low-performance (and thus low-power) states when the performance demand is not at the peak level by simultaneously scaling down their supply voltage and processing frequency [35].

Although both redundancy-based fault tolerance [6,17] and DPM/DVFS-based energy management schemes [35,52] have been independently studied extensively, the co-management of system

* Corresponding author.
E-mail address: dakai.zhu@utsa.edu (D. Zhu).

http://dx.doi.org/10.1016/j.sysarc.2017.06.008
1383-7621/© 2017 Elsevier B.V. All rights reserved.

reliability and energy consumption has caught researchers' attention only very recently [14,31,38]. Note that, fault tolerance and energy efficiency are normally conflicting design objectives in computing systems since redundancy generally results in increased energy consumption [2]. Moreover, recent studies show that DVFS has a negative effect on system reliability due to significantly increased transient fault rates at low supply voltages [11,15,49]. With this intriguing interplay between fault tolerance and energy efficiency, it becomes imperative to develop effective techniques that can address both dimensions while guaranteeing the timeliness of real-time tasks.

By taking the negative effects of DVFS on transient fault rates into consideration, a series of *reliability-aware power-management (RAPM)* schemes have been studied for various real-time task models based on the backward recovery technique [13,19,29,34,45–48]. Basically, RAPM exploits system slack (i.e., *temporal redundancy*) for both reliability preservation and energy savings. RAPM ensures to schedule a recovery task before scaling down the processing frequency of any task, using the remaining slack time. By executing the recovery task at the maximum frequency, RAPM can achieve a desired system reliability level even if the task with scaled frequency incurs transient faults [29,45,48]. Although RAPM can guarantee system reliability with respect to transient faults (which were shown to be more common [26]), it does not offer provisions for tolerating permanent faults.

With the objective of tolerating a single permanent fault while guaranteeing system reliability with respect to transient faults, the *Standby-Sparing (SS)* schemes were recently studied for both aperiodic [12] and periodic tasks [22,23] running on a dual-processor system based on the *Primary/Backup (PB)* fault-tolerance technique. Essentially, the SS schemes schedule primary and backup tasks *separately* on the primary and secondary processors, respectively, to tolerate one permanent fault. Note that, to improve system efficiency and reduce execution overhead (thus to save energy), the backup tasks are normally cancelled as soon as their corresponding primary tasks complete successfully [6] and should be scheduled as late as possible [38]. Hence, for energy-efficiency (and reliability preservation), the SS schemes execute primary tasks early at scaled frequency while backup tasks at the maximum frequency at their latest times on their dedicated processors, respectively [12,22,23]. Although the SS schemes can effectively tolerate a single permanent fault with some energy savings, the available slack time on the secondary processor is not efficiently utilized with the adopted DPM technique.

Instead of dedicating one processor for backup tasks, the primary and backup tasks can be allocated in a mixed manner on both processors and all available slack time can be exploited by the DVFS technique for better energy savings [20]. Here, each processor is allocated a mixed set of primary and backup tasks where primary tasks exploit the slack time and run at a scaled frequency with DVFS. Moreover, the tasks are scheduled with the *preference-oriented earliest-deadline (POED)* scheduler, which can differentiate them and execute primary tasks *as soon as possible (ASAP)* while backup tasks *as late as possible (ALAP)* [18], for better energy savings. The same idea of allocating tasks in a mixed manner has been exploited in the *Secondary Execution Time Shifting (SETS)* offline scheduling heuristic for saving energy while tolerating faults for periodic tasks running on multiprocessor systems [38]. However, the aforementioned studies either focused on dual-processor systems [12,20,22,23] or did not consider the more effective DVFS power management technique [38].

To the best of our knowledge, there is no existing work that address how to effectively schedule periodic real-time tasks in multiprocessor systems to save energy with both DPM/DVFS techniques while tolerating a single permanent fault and preserving system reliability with respect to transient faults. By extending our preliminary study [21], we focus on such a problem in this paper and propose several *energy-efficient fault-tolerance (EEFT)* schemes. In particular, the contributions of this work are summarized as follows:

- First, we study two Standby-Sparing (SS) based EEFT schemes: *Paired-SS* organizes processors as groups of two (i.e., *pairs*) and adopts the existing SS scheme [12] for each processor pair; *Generalized-SS* divides processors into *primary* and *secondary* processor groups (of potentially different sizes) and then schedules primary (backup) tasks on the primary (secondary) processors under the *partitioned-EDF* (*partitioned-EDL*) with DVFS (DPM) to save energy [21];
- Second, by allocating primary and backup tasks in a mixed manner on all processors to better utilize their slack time for more energy savings, we propose two novel EEFT schemes based on the POED scheduling algorithm; Here, once primary tasks are partitioned to *all* processors (e.g., according to the Worst-Fit-Decreasing heuristic), backup tasks can be allocated to processors following either *Cyclic* or *Mixed* approach;
- Finally, the proposed EEFT schemes are evaluated through extensive simulations and the results show their effectiveness on energy savings.

The remainder of this paper is organized as follows. Section 2 reviews the closely-related work. Section 3 presents system models and states the assumptions of this work. The Standby-Sparing based schemes are discussed in Section 4 and the POED-based schemes are investigated in Section 5. The evaluation results are presented and discussed in Section 6 and Section 7 concludes the paper.

2. Closely related work

Aiming at tolerating a given number of transient faults in a real-time application, Melhem et al. [31] derived the optimal number of checkpoints, uniformly or non-uniformly distributed, to achieve the minimum energy consumption for a duplex system (where two hardware units are used to run the same software concurrently for fault detection) with the DVFS power management technique [31]. Assuming that transient faults follow a Poisson distribution with a constant arrival rate, Zhang et al. studied an adaptive checkpointing scheme to tolerate a fixed number of transient faults during the execution of a real-time task [42]. The adaptive checkpointing scheme was extended to a set of periodic tasks on a single processor system with the EDF scheduler [44]. In [43], the authors further considered the cases where faults may occur within checkpoints. Following a similar idea and considering a fixed-priority RMS algorithm, Wei *et al.* studied an efficient online scheme to minimize energy consumption by considering the run-time behaviors of tasks and fault occurrences while satisfying tasks' timing constraints [39]. In [40], the authors extended the study to multiprocessor real-time systems.

Elnozahy et al. studied an Optimistic-TMR (OTMR) scheme to reduce the energy consumption in a Triple Modular Redundancy (TMR) system in [14]. OTMR allows one processing unit to run at a scaled frequency with DVFS provided that it can catch up and finish the computation before the deadline if a fault does occur on other two units. The optimal frequency settings for OTMR was explored in [50]. For independent service requests, Zhu et al. studied the optimal redundant configuration for server processors to tolerate a given number of transient faults [51]. Izosimov et al. [27] studied an optimization problem for mapping a set of tasks with reliability constraints, timing constraints and precedence relations to processors for determining the appropriate fault tolerance policy (re-execution or replication) for the tasks [27].

However, despite the effectiveness of DVFS on reducing energy consumption, recent studies showed that it has a negative effect on system reliability due to the significantly increased transient fault rates at low supply voltages [15]. In particular, an exponential fault rate model with scaled voltage was proposed in [49]. Taking such negative effects of DVFS into consideration, Zhu studied a *Reliability-Aware Power Management (RAPM)* scheme that schedule a recovery task before exploiting the remaining slack time to scale down the execution of the primary task [47]. Here, to preserve system reliability with respect to transient faults, the recovery task is executed at the maximum frequency only if transient faults cause an error during the primary task's execution. Later, the RAPM scheme was extended for periodic real-time tasks [48].

To address the conservativeness of RAPM that schedules an individual recovery task for each task running at scaled frequency, Zhao et al. [46] studied the *Shared-Recovery (SHR)* technique [46], where several scaled tasks can share a recovery task to leave more slack for DVFS and save more energy. To achieve an arbitrary system-level target reliability, SHR has been further extended to the *generalized shared recovery (GSHR)* where a small number of recovery tasks are shared by all the tasks [45]. A similar study was also reported recently in [29]. Global scheduling based RAPM schemes for both independent [34] and dependent [19] real-time tasks running on multiprocessor systems were studied as well.

Moreover, based on the exponential fault rate model developed in [49], Ejlali et al. [13] studied a number of schemes that combine the information about hardware resources and temporal redundancy to save energy and to preserve system reliability [13]. Considering dependent tasks represented by directed acyclic graphs (DAGs), Pop et al. proposed a novel framework by studying the energy and reliability trade-offs for distributed heterogeneous embedded systems [32]. By employing a feedback controller to track the overall miss ratio of tasks in soft real-time systems, Sridharan et al. [36] proposed a reliability-aware energy management algorithm to minimize the system energy consumption while still preserving the overall system reliability. Dabiri et al. [10] studied the problem of assigning frequency and supply voltage to tasks for energy minimization subject to reliability as well as timing constraints [10]. For a real-time application running a dual-processor system, Aminzadeh and Ejlali [2] performed a comparative study of different DVFS and DPM schemes to tolerate a given number of transient faults [2]. Although the above work can preserve system reliability with respect to transient faults, there is no provision for permanent faults.

Based on the *primary/backup* technique, Bertossi et al. [6] studied several fixed-priority RMS scheduling algorithms for periodic real-time tasks to tolerate a given number of permanent faults [6], where the goal is to improve system resource utilization through *backup deallocation*. In [5], the authors further proposed the backup phasing delay technique to reduce the overlapped executions between the primary and backup tasks. However, these work did not consider energy management. Based on the EDF scheduling, Unsal et al. studied an offline Secondary Execution Time Shifting (SETS) heuristic for a set of independent periodic real-time tasks running on multiprocessor systems [38]. Here, to obtain an energy-efficient static schedule within the least common multiple (LCM) of tasks' periods, SETS iteratively delays the release time of backup tasks to reduce the overlapped executions with their corresponding primary tasks and thus to reduce system energy consumption, but without exploiting the more effective DVFS technique.

To tolerate a single permanent fault while taking transient faults into consideration, Ejlali et al. [12] investigated a *Standby-Sparing (SS)* scheme to save energy for dependent and aperiodic real-time tasks running on a dual-processor system [12]. SS executes primary tasks with DVFS on one processor (denoted as the *primary* processor) at their earliest times while backup tasks with DPM on another (*spare*) processor at their latest times to reduce their overlapped executions and thus to save more energy. The work was extended later with a light-weight feedback system for better energy savings [37]. With the same idea of *separating* tasks on the two processors, Haque et al. extended standby-sparing to a more practical periodic task model based on the earliest deadline schedulers [22], where primary and backup tasks are scheduled according to EDF with DVFS and EDL [8] with DPM on their dedicated processors, respectively, to save energy. The fixed-priority based standby-sparing scheme was further studied in [23]. More recently, for multicore systems with energy harvesting, Xiang and Pasricha [41] proposed a hybrid design-time/run-time framework for resource allocation that takes into consideration of variations in solar radiance and execution time, transient faults, and permanent faults due to aging effects [41].

Observing the inefficient usage of slack time with DPM on the spare processor, we proposed to schedule a mixed set of primary and backup copies of different tasks on both processors [20]. Based on the *Preference-Oriented Earliest Deadline (POED)* scheduling algorithm [18], all available slack time on both processors can be utilized to scale down primary tasks with DVFS for better energy savings.

In this paper, we focus on the problem of how to effectively schedule periodic real-time tasks on a multiprocessor system to save energy with both DPM/DVFS techniques while tolerating a single permanent fault and preserving system reliability with respect to transient faults, which is different from all existing work. Specifically, we generalize Standby-Sparing and POED-based schemes to the settings with multiprocessor systems.

3. Preliminaries and system models

3.1. System, task and power models

We consider a homogeneous m-processor shared-memory system. As power management features are common in modern processors [1,9], we assume that all processors adopted in the system have the *dynamic voltage and frequency scaling (DVFS)* capability, which allows them to operate at one of L discrete frequency (and voltage) levels ($F_1 < F_2 < \ldots < F_L$). We consider normalized frequencies and assume that the maximum frequency is $F^{\max} = F_L = 1.0$.

The system has a set of n periodic real-time tasks $\Gamma = \{T_1, \ldots, T_n\}$, where each task T_i is represented as a tuple (c_i, p_i). Here c_i is T_i's worst-case execution time (WCET) under the maximum available processor frequency $F^{\max}$ and, p_i is its period. The tasks are assumed to have implicit deadlines. That is, the jth task instance (or *job*) of T_i, denoted as $T_{i,j}$, arrives at time $(j-1)\cdot p_i$ and needs to complete its execution by its deadline at $j \cdot p_i$. Note that, a task has only one active task instance at any time. Hence, when there is no ambiguity, we use T_i to represent both the task and its current task instance. The utilization of a task T_i is defined as $u_i = \frac{c_i}{p_i}$. The system utilization of a given task set is further defined as the summation of all tasks' utilization: $U(\Gamma) = \sum_{T_i \in \Gamma} u_i$.

The tasks are assumed to be *independent* and share no resource other than the processors. Moreover, we do not consider the effects of memory access on tasks' execution time, which is assumed to scale linearly with the operation frequency of its processor. That is, if task T_i's processor operates at frequency F_k, the WCET of T_i will be $\frac{c_i}{F_k}$. It is possible to model the memory effects with a frequency-independent portion in the execution time [3]. However, it is beyond the scope of this paper and exploring this direction will be left for our future work.

With the shrinking technology size, the static and leakage power increases in a faster pace when compared to that of dynamic power [28]. Hence, it becomes more important to manage

power consumption at the system-level with all power consuming components being considered [3,25]. Although more precise power models at micro-architecture level have been studied [2,30], we adopt in this work a simple system-level power model to simplify the analysis and discussions, which has also been widely exploited in recent studies [29,34,48]. Specifically, for a system with m processors (that operate at $f_1, \ldots, f_m$, respectively), its power consumption can be expressed as:

$$P(f_1, \ldots, f_m) = P_s + \sum_{i=1}^{m} \hbar_i (P_{ind} + C_{ef} \cdot f_i^k) \quad (1)$$

where P_s stands for system static power, which can be removed only by powering off the whole system. However, due to the prohibitive overhead of turning off and on the system [14] in periodic real-time execution settings, we assume that the system is in *on* state at all times and that P_s is always consumed. That is, we will focus on the energy consumption related to the system active power, which is given by the second item in the above equation.

On each processor, if it is actively executing tasks, two components of active power are consumed: the *frequency-independent* active power P_{ind} (which is assumed to be the same for all processors) and the frequency-dependent active power (which depends on the system-dependent constants C_{ef} and k, as well as the processor's frequency f_i). That is, if the ith processor is active, we have $\hbar_i = 1$. Otherwise, if there is no ready task on the ith processor, it can switch to the *sleep* state through the dynamic power management (DPM) and does not consume any active power (i.e., $\hbar_i = 0$).

Considering the fact that modern processors can switch to sleep states in a few cycles [1,9], we assume that the time overhead for a processor to enter/exit its sleep state is negligible. Moreover, to simplify the discussion, the overhead for frequency (and voltage) changes under DVFS is also assumed to be included into tasks' WCETs or can be incorporated with the slack reservation mechanism [48].

From the above system-level power model, an energy-efficient frequency, $F_{ee} = \sqrt[k]{\frac{P_{ind}}{C_{ef} \cdot (k-1)}}$, can be derived, below which DVFS consumes more energy to execute a task [34]. In what follows, we assume that all available frequency levels are energy-efficient and hence, $F_{ee} \leq F_1$ holds.

3.2. Fault and recovery models

During the operation of a real-time system, different faults may occur due to hardware failure, software errors or electromagnetic interference. While transient faults can be tolerated with *temporal* redundancy, permanent faults can only be tolerated through *modular/hardware* redundancy. With the scaled technology size [15] and widely-adopted DVFS technique, modern computing devices are more susceptible to transient faults [11]. In particular, as supply voltage is reduced with DVFS to save energy, the rate of transient faults may increase exponentially [49]. Moreover, although the occurrence of permanent faults is very rare, a comprehensive framework should have provisions for both transient and permanent faults in a safety-critical multiprocessor real-time system.

With the objective of tolerating both transient and permanent faults, we adopt the *Primary/Backup (PB)* fault-tolerance technique in this work. That is, for each task T_i, there is a periodic *backup* task B_i. To distinguish between them, we occasionally use the term *primary* (or *main*) task to refer to T_i. To ensure that there is a proper backup for every task instance of T_i, we assume that B_i has the same timing parameters[1] (i.e., c_i and p_i) as T_i. Hence, in addition to the original primary task set Γ, we have a set Γ^B of backup tasks that have to be properly scheduled.

As in most existing fault tolerance work, we assume that fault detection mechanisms are available in the system and the detection overhead has been incorporated into the WCETs of tasks [2]. Specifically, the soft errors caused by transient faults are detected at the end of a task's execution through the *sanity* (or *consistency*) checks (e.g., parity or signature checks) [33]. For permanent faults, we assume the failure-stop model and a faulty processor can be detected by other working ones at the earliest completion time of a task [33].

Problem description: On a multiprocessor system where both DVFS and DPM techniques are available for energy management, how one should efficiently schedule the main and backup tasks to maximize the energy savings under the constraints of (a) tolerating a single permanent fault; and (b) preserving system reliability with respect to transient faults (in the absence of permanent faults).

The backup tasks adopted in this work have dual purposes. First, with one backup for each main task, the system is inherently robust to a single permanent fault provided that the main and backup copies of the same task are scheduled on different processors [6]. The second objective of having backup tasks is, in the absence of permanent faults, to preserve system reliability[2] with respect to transient faults when the execution of primary tasks is scaled down with DVFS to save energy. Therefore, by considering the negative effects of DVFS on transient fault rates [49], backup tasks are assumed to be executed at the maximum frequency [12,48].

4. Standby-Sparing for multiprocessor

4.1. An example with a three-processor system

When there are more (i.e. > 2) processors in a system, a natural question to ask would be: *"how to configure such processors for better energy efficiency?"* We can either have additional primary processors to execute main tasks at further reduced frequencies or have more secondary processors to further delay the execution of backup tasks. Clearly, this is not a trivial problem considering the intriguing interplay between the scaled frequency of main tasks and the amount of overlapped execution with their backup tasks.

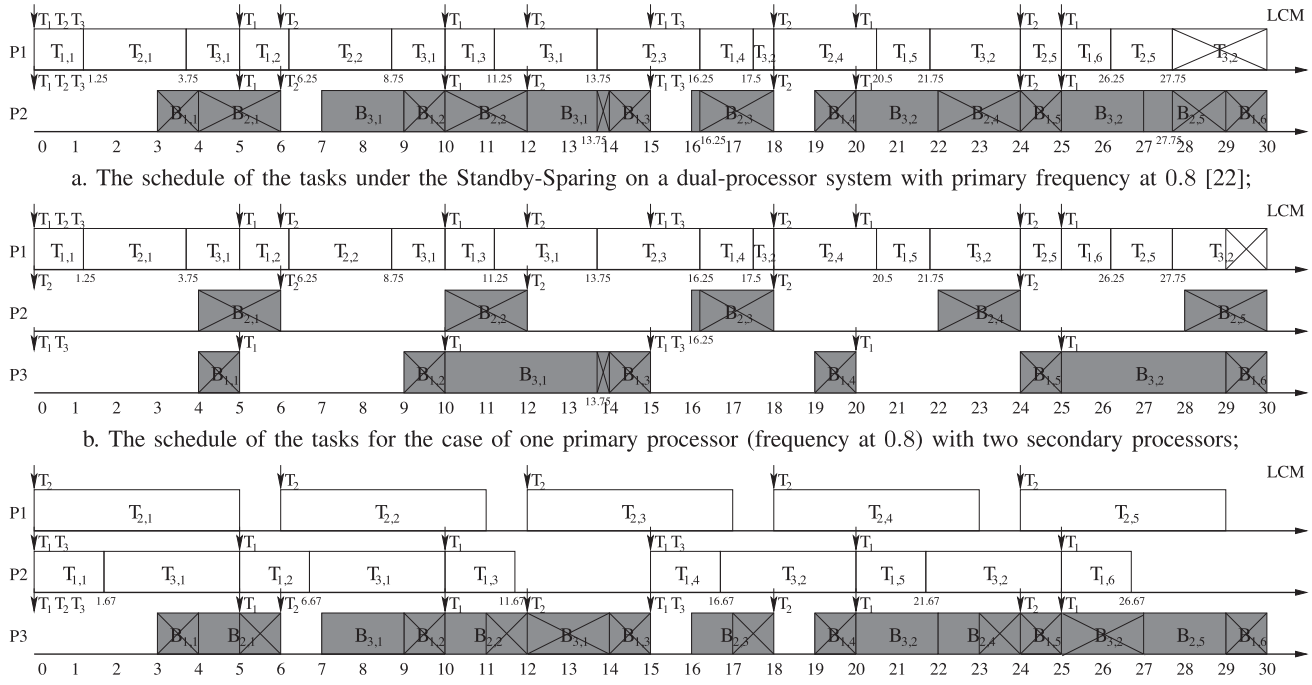
Before presenting the solution for the general problem for multiprocessor systems, we first investigate the simple case of a three-processor system. Here, we have two options for the configuration of the processors: (a) one primary and two secondary processors (denoted as "X1Y2"); and (b) two primary and one secondary processor (denoted as "X2Y1").

A motivational example: Consider a task set with three periodic real-time tasks $\Gamma = \{T_1(1, 5), T_2(2, 6), T_3(4, 15)\}$. We can easily find that the system utilization is $U = 0.8$ and the *least common multiple (LCM)* of tasks' periods is $LCM = 30$. Suppose that the processors have four discrete (normalized) frequency levels {0.4, 0.6, 0.8, 1.0}.

Fig. 1(a) first shows the tasks' schedule on a dual-processor system with the Standby-Sparing technique within LCM [22]. The primary processor executes the main tasks under *Earliest Deadline First (EDF)* at a scaled frequency of 0.8 while the secondary processor schedules the backup tasks with *Earliest Deadline Latest (EDL)* [8] policy for energy savings. By assuming $P_{ind} = 0.01$, $C_{ef} = 1$ and $k = 3$ in the power model, we can find the active energy consumption within LCM is $E_{SS-SPM} = 27.2$ when all tasks take their WCETs and there is no fault at run-time. The executions of

[1] Note that, as long as B_i's WCET is no more than that of T_i (i.e., B_i can be either a reduced version or the replication of T_i), the proposed schemes can guarantee system reliability with respect to transient faults [12,29,48].

[2] Higher levels of system reliability can be achieved with additional replicated copies of tasks [29,45]. However, exploring this direction is beyond the scope of this paper and will be investigated in our future work.



a. The schedule of the tasks under the Standby-Sparing on a dual-processor system with primary frequency at 0.8 [22];

b. The schedule of the tasks for the case of one primary processor (frequency at 0.8) with two secondary processors;

c. The schedule of the tasks for the case of two primary processors (frequencies at 0.4 and 0.6) with one secondary processor;

Fig. 1. A set of three tasks $T_1(1, 5)$, $T_2(2, 6)$ and $T_3(4, 15)$ running on a three-processor system.

most backup tasks are cancelled, which are marked with an 'X' in the figure.

For the X1Y2 configuration of a three-processor system, where the extra processor is used as an additional secondary, the schedule of tasks is shown in Fig. 1(b) The primary processor $\mathcal{P}_1$ still runs at the frequency 0.8 to execute the main tasks. However, backup tasks can be re-allocated, where backup task B_2 is allocated to processor $\mathcal{P}_2$ and B_1 and B_3 to processor $\mathcal{P}_3$, to further delay and reduce their overlapped executions. It turns out that, although the overlapped executions can be reduced slightly, the additional energy consumption from the frequency-independent active power (i.e., P_{ind}) of the extra processor overshadows such benefits and leads to the total active energy consumption of $E_{X1Y2} = 27.45$, which is slightly more than that of the traditional Standby-Sparing scheme for the dual-processor system.

However, when the extra processor is utilized as an additional primary processor, Fig. 1c shows the schedule with the X2Y1 configuration. Here, the main task T_2 is allocated to processor $\mathcal{P}_1$ and other two tasks (T_1 and T_3) to $\mathcal{P}_2$, which can run at the scaled frequencies of 0.4 and 0.6, respectively. The total active energy consumption under this configuration can be found as $E_{X2Y1} = 23.37$. Compared to that of the dual-processor system with the traditional Standby-Sparing scheme, this gives a 14% improvement.

4.2. Standby-Sparing based schemes

From the above example, we can see that different configurations of primary and secondary processors can have important effects on the energy efficiency of a multiprocessor system. Following the principles and extending the ideas of the traditional Standby-Sparing scheme [22], we propose in what follows the *Paired Standby-Sparing* (Section 4.2.1) and *Generalized Standby-Sparing* (Section 4.2.2) schemes for periodic real-time tasks running on multiprocessor systems.

4.2.1. Paired Standby-Sparing (P-SS)

Considering the fact that the traditional Standby-Sparing scheme was designed for dual-processor systems, a simple and straightforward approach is to first organize the processors in a system as groups of two (i.e., *pairs*). Then, the existing Standby-Sparing scheme can be applied directly to each pair of processors after partitioning (main and backup) tasks to the processor pairs appropriately, which is thus named as the *Paired Standby-Sparing (P-SS)* scheme.

From the results in [22], we know that different system utilizations of tasks have a great impact on the energy efficiency of a dual-processor system under the Standby-Sparing scheme. The reason is that, both the scaled frequency for the main tasks on the primary processor and the delayed execution of backup tasks on the secondary processor depend heavily on system loads. When the system utilization of a given task set is high, the Standby-Sparing scheme could perform quite poorly due to higher execution frequency for main tasks and the increased amount of overlapped execution between the main and backup tasks. On the other hand, once the scaled execution frequency of the main tasks reduces to the minimum (available) energy-efficient frequency, additional energy savings are rather limited with further reduced system loads [22].

Therefore, the *key* factor for the energy efficiency of a multiprocessor system under P-SS will be the *mapping* of tasks to processor pairs. However, it is well-known that the problem of finding a *feasible* mapping of a given set of periodic real-time tasks in a multiprocessor system is NP-hard. Therefore, finding the optimal mapping of (main and backup) tasks among the processor pairs in P-SS to minimize the system energy consumption is NP-hard as well. Note that, without the consideration of fault tolerance, a balanced workload distribution has been shown to have the best energy efficiency for tasks running on a multiprocessor system [4]. Hence, following this intuition and considering its inherent ability to obtain a load-balanced mapping, we adopt the *Worst-Fit Decreasing (WFD)* heuristic in P-SS when mapping (main and backup) tasks to the processor pairs.

Note that, to apply the traditional Standby-Sparing within each processor pair, the main and backup of the same task (e.g., T_i and B_i) have to be mapped to the same pair of processors. Therefore, in P-SS, we can first map the main tasks in Γ to the processor pairs according to the WFD heuristic. That is, each processor pair will be allocated a subset Γ_q of the main tasks, where $1 \le q \le \lfloor \frac{m}{2} \rfloor$ as there are at most $\lfloor \frac{m}{2} \rfloor$ processor pairs for a system with m processors. Then, for each backup task B_i, it will be allocated to the processor pair that contains the corresponding main task T_i,

With the Standby-Sparing scheme being adopted within each processor pair, the main and backup tasks are scheduled under EDF and EDL on the primary and secondary processors, respectively [22]. Recall that backup tasks have the same timing parameters (i.e., utilizations) as their main tasks. Therefore, the resulting WFD mapping $\{\Gamma_q\}$ (and corresponding $\{\Gamma_q^B\}$) is feasible if there are $U(\Gamma_q) \le 1$ $(1 \le q \le \lfloor \frac{m}{2} \rfloor)$.

Once the feasible WFD mapping is obtained, the processor pairs under P-SS will operate independently. Although each processor pair acting as a Standby-Sparing system can tolerate one permanent fault [22], it is possible for multiple permanent faults hit both processors in one pair. Hence, with each task having one backup, P-SS can only tolerate a single permanent fault in the worst case scenario. However, once the processor affected by permanent fault(s) is identified, additional permanent faults could be tolerated by re-configuring the remaining $(m-1)$ processors and re-map the tasks.

4.2.2. Generalized Standby-Sparing (G-SS)

From the example system with three processors (Section 4.1), we have seen that having two primary processors to execute the main tasks while sharing one secondary processor among the backup tasks can lead to better energy efficiency. Following this principle and generalizing the idea of Standby-Sparing, we propose the *Generalized Standby-Sparing (G-SS)* scheme, which organizes the m processors of the system into two groups: the primary group of X processors and the secondary group of Y processors, where $m = X + Y$. Then, the main and backup tasks are *separately* scheduled on the processors in the primary and secondary groups, respectively.

Considering the fact that the EDF/EDL schedulers are exploited in the P-SS scheme and the simplicity of partitioned scheduling, we adopt the partitioned-EDF and partitioned-EDL for G-SS to schedule the main and backup tasks, respectively. Hence, for a given (X, Y)-configuration of the processors, Algorithm 1 summarizes the major steps of G-SS.

Algorithm 1 G-SS for a given (X, Y)-configuration.

1: **Input:** task sets Γ and Γ^B; X and $Y(= m - X)$;
2: **Output:** Scaled frequencies for primary processors and EDL schedules for secondary processors;
3: Find the (X, Y) WFD partitions of Γ and Γ^B:
4: $\Pi(X) = \{\Gamma_1, \cdots, \Gamma_X\}$ and $\Pi^B(Y) = \{\Gamma_1^B, \cdots, \Gamma_Y^B\}$;
5: **if** $(\forall i, U(\Gamma_i) \le 1$ and $\forall j, U(\Gamma_j^B) \le 1)$ **then**
6: //Suppose the first X processors are primary processors
7: **for** (each primary processor $\mathcal{P}_x$: $x = 1 \to X$) **do**
8: $f_x = \min\{F_i | F_i \ge U(\Gamma_x), i = 1, \ldots, L\}$;
9: **end for**
10: **for** (each secondary processor $\mathcal{P}_y$: $y = 1 \to Y$) **do**
11: Generate the offline EDL schedule for tasks in Γ_y^B;
12: **end for**
13: **end if**

First, the main and backup tasks are partitioned among the X primary and Y secondary processors, respectively (lines 3 and 4). Again, to obtain the mappings with balanced-workload for better energy savings, the WFD heuristic is adopted [4]. Then, the schedulabilities of the resulting WFD mappings for both the main and backup tasks under the EDF and EDL schedulers on the primary and secondary processors, respectively, are examined (line 5).

If any processor is overloaded with the resulting mappings $\Pi(X)$ and $\Pi^B(Y)$, we say that the (X, Y)-configuration is *not feasible*. Otherwise, to save energy, the scaled frequency for each primary processor to execute its main tasks is determined (lines 7 and 8); in addition, assuming that backup tasks run at the maximum frequency, the EDL schedule for each secondary processor is generated offline (lines 10 and 11).

Since all backup tasks run on processors that are different from their corresponding main tasks, G-SS is able to tolerate a single permanent fault. Moreover, the system reliability with respect to transient faults can also be preserved since all backup tasks are assumed to run at $F^{\max}$. Note that, as in the traditional Standby-Sparing scheme [22], if a main (or backup) task completes successfully on one processor at runtime, the related processor will be notified to cancel the execution of the corresponding backup (or main) task for energy savings.

It is clear that different configurations of the processors in G-SS have a great impact on the energy efficiency of a multiprocessor system. For the special case with the same number of primary and secondary processors (i.e., $X = Y$), we can find that G-SS will be effectively reduced to P-SS since they adopt the same WFD mapping heuristic and the backup tasks have the same timing parameters as their corresponding main tasks. However, for the configurations that have different numbers of primary and secondary processors (i.e., $X \ne Y$), it is very likely that the backup tasks will be mapped to different secondary processors in G-SS even if their main tasks are mapped to the same primary processor. Due to such implications, it is quite difficult to identify the overlapped execution regions between the main and backup tasks in the EDF and EDL schedules on different processors, which makes it almost impossible to find the optimal configuration of processors for G-SS to minimize energy consumption analytically.

Energy-efficient configuration: For a given task set Γ running on a m-processor system, the major steps for an iterative algorithm to find out the energy-efficient processor configuration for G-SS to minimize the system energy consumption are shown in Algorithm 2. Note that, with the system utilization of $U(\Gamma)$ for a task set Γ, the minimum number of required primary processors for the tasks to be schedulable under partitioned-EDF can be obtained as $X^{\min} = \lceil U(\Gamma) \rceil$. $X^{\min}$ also gives the minimum number of required secondary processors. Thus, the maximum number of primary processors can be found as $X^{\max} = m - X^{\min}$ (line 3).

Algorithm 2 Find the energy-efficient configuration for G-SS.

1: **Input:** task sets Γ and Γ^B; m (number of processors);
2: **Output:** the energy-efficient processor configuration (i.e., X^{opt}) for G-SS to minimize energy consumption;
3: $X^{\min} = \lceil U(\Gamma) \rceil$; $X^{\max} = m - X^{\min}$;
4: $E^{\min} = \infty$; $X^{opt} = -1$; //initialization
5: **for** $(X = X^{\min} \to X^{\max})$ **do**
6: $Y = m - X$; //number of secondary processors
7: **if** (Γ is schedulable under G-SS with X/Y) **then**
8: Get $E_{G-SS}(X, Y)$ from emulation in LCM;
9: **if** $(E^{\min} > E_{G-SS}(X, Y))$ **then**
10: $X^{opt} = X$;
11: **end if**
12: **end if**
13: **end for**

For each possible (X, Y)-configuration of the processors, the schedulability of the given task set Γ under G-SS can be checked

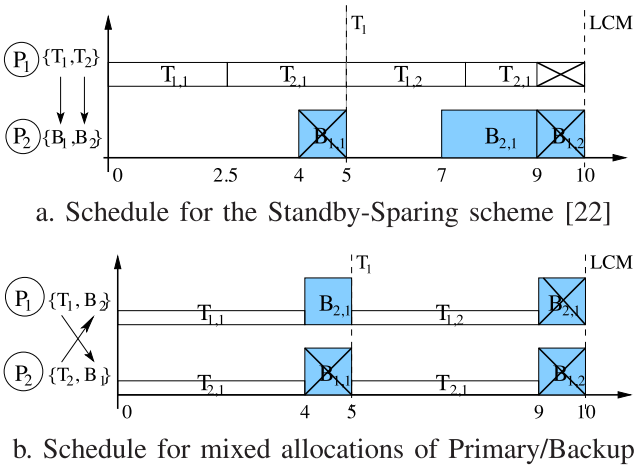


Fig. 2. An example of two tasks $T_1 = (1, 5)$ and $T_2 = (2, 10)$ running on a dual-processor system.

using Algorithm 1 (lines 5–7). If Γ is schedulable, the system energy consumption under G-SS can be obtained from the emulated execution of the tasks within LCM (line 8). During such emulations, we assume that tasks take their WCETs and no fault occurs. Finally, searching through all feasible configurations of the processors, the most energy-efficient (X^{opt}, Y^{opt})-configuration with the lowest system energy consumption can be found out (lines 9 and 10). As shown in Section 6.1, such a configuration for G-SS normally has more primary processors and can lead to better energy efficiency when compared to that of the P-SS scheme.

5. Mixing primary/backup tasks

The separation of main and backup tasks on their dedicated processors simplifies the scheduling algorithm on each processor for the SS-based schemes. However, since backup tasks need to run at the maximum frequency for reliability preservation, the available slack time on secondary processors can only be used to idle processors with the DPM technique to save energy. As illustrated in the following example of a dual-processor system, better energy savings can be obtained if the main and backup tasks are allocated in a mixed manner on both processors [20], which can more efficiently utilize all available slack time with the DVFS technique.

5.1. Inefficient slack usage in Standby-Sparing

Consider a dual-processor system with two periodic tasks $T_1 = (1, 5)$ and $T_2 = (2, 10)$. The schedule within the LCM of tasks' periods under the Standby-Sparing scheme is shown in Fig. 2(a) Here, the main tasks T_1 and T_2 are executed at the scaled frequency of 0.4 on the primary processor under EDF, while the backup tasks B_1 and B_2 are scheduled on the secondary processor under EDL [22]. Clearly, as B_1 and B_2 are required to run at the maximum frequency for reliability preservation, the slack time on the secondary processor can only be exploited by DPM to idle the processor.

However, it is well-known that slack time can be more efficiently utilized by the DVFS technique [35,52]. Therefore, instead of dedicating one processor for backup tasks, we can allocate the main and backup tasks to both processors in a *mixed* manner as shown in Fig. 2b. Here, T_1 and B_2 are allocated to the first processor while T_2 and B_1 to the second processor. Hence, each processor can utilize its slack time for its main task. It turns out that, with DVFS, both T_1 and T_2 can be executed at the scaled frequency of 0.25.

Suppose that tasks take their WCETs and no fault occurs during tasks' executions. When tasks are executed according to the schedule within the LCM as shown in Fig. 2(b), most executions of backup tasks will be cancelled (marked with 'X'). Hence, when compared to the case of the Standby-Sparing schedule in Fig. 2(a), about 20% more energy savings can be obtained under the new scheme with mixed allocations of main and backup tasks on both processors.

However, we should point out that it is not trivial to obtain such a schedule as in Fig. 2(b), which is neither an EDF nor EDL schedule. From the figure, we can see that, to obtain more energy savings, the main tasks on each processor are executed at their earliest times while the backup tasks are delayed as much as possible (without causing any deadline miss). To efficiently generate such schedules, in what follows, we first review the basic ideas of the *preference-oriented earliest deadline (POED)* scheduling algorithm [18], which forms the foundation of the novel energy-efficient fault-tolerance schemes with mixed allocations of main and backup tasks.

5.1.1. A preference-oriented scheduling algorithm

Basically, POED is a dynamic-priority based scheduler for periodic real-time tasks running on a single processor system. However, as opposed to the conventional earliest-deadline schedulers, such as EDF and EDL [8] (which treat all tasks uniformly and schedule them at their earliest and latest times, respectively), POED can distinguish different execution preferences of tasks, which can be either *as soon as possible (ASAP)* or *as late as possible (ALAP)* [18].

To incorporate such execution preferences of tasks, POED follows two principles when making scheduling decisions [18]. First, even if an ASAP task has a later deadline than that of an ALAP task, the ASAP task should be executed before the ALAP task if it is possible to do so without causing any deadline miss. Second, the execution of ALAP tasks should be delayed as much as possible given that it does not cause any deadline miss for both current and future tasks.

Given these two principles, at any scheduling event (such as the arrival or completion of a task, or a timer interrupt), the basic steps of the POED scheduler can be summarized as follows. For cases where the ready task with the highest priority (i.e., earliest deadline) has ASAP preference, POED will execute the task normally as in EDF. However, in case an ALAP task has the earliest deadline, POED will focus on a *look-ahead interval* from the invocation time to the earliest deadline of an ASAP task. All (current and future arrival) tasks within this interval will be considered to see whether it is safe to delay the ALAP task's execution and if yes, for how long can it be delayed. We have shown that POED can guarantee to meet all tasks' deadlines when scheduling them according to their preferences. In particular, we have the following theorem regarding to the schedulability of a task set under POED. Interested readers can refer to [18] for the detailed analysis.

Theorem 1 (POED Schedulability [18])**.** *For a set Γ of periodic tasks with either ASAP or ALAP preferences, no task will miss its deadline under POED if $U(\Gamma) \leq 1$.*

Therefore, with the POED scheduler, the main tasks (i.e., T_1 and T_2) in the above example will have ASAP preference while the backup tasks (i.e., B_1 and B_2) have ALAP preference on their respective processors. Moreover, when the scaled frequency for the main tasks is 0.25, the *inflated* system utilization is exactly 1 on both processors. Hence, from Theorem 1, the mixed sets of main and backup tasks on both processors can be successfully scheduled under POED, which results in the schedule as shown in Fig. 2(b).

5.2. POED-based EEFT schemes

There are two reasons for the significant energy savings when a mixed set of main and backup tasks are allocated to each processor and scheduled under POED. First, the slack time on all processors can be efficiently exploited by their main tasks with the DVFS technique. Second, with the POED scheduler, most executions of backup tasks can be effectively cancelled at runtime as such executions are delayed as much as possible while the corresponding main tasks are executed (on another processor) at their earliest times. By generalizing these ideas, the major steps of the POED-based energy-efficient fault-tolerance (EEFT) schemes for multiprocessor systems can be summarized in Algorithm 3.

Algorithm 3 Major steps of POED-based EEFT schemes.

1: **Input:** task sets Γ and Γ^B; number of processors m;
2: **Step 1:** Allocate main tasks in Γ to m processors;
3: Suppose the WFD partition is $\Pi = \{\Gamma_1, \ldots, \Gamma_m\}$;
4: **Step 2:** Allocate backup tasks in Γ^B to all processors;
5: Suppose backup partition is $\Pi^B = \{\Gamma_1^B, \ldots, \Gamma_m^B\}$;
6: **Step 3:** Calculate scaled frequencies for main tasks;
7: **for** $(i : 1 \rightarrow m)$ **do**
8: $f_i = \min\{F_x | F_x \geq \frac{U(\Gamma_i)}{1-U(\Gamma_i^B)}, x = 1, \ldots, L\}$;
9: Assign f_i to the main tasks in Γ_i;
10: Assign $f^{\max} = F_L$ to the backup tasks in Γ_i^B;
11: **Step 4:** Execute tasks on each processor under POED;
12: **for** $(i : 1 \rightarrow m)$ **do**
13: Assign ASAP preference to main tasks in Γ_i;
14: Assign ALAP preference to backup tasks in Γ_i^B;
15: Execute Γ_i and Γ_i^B on $\mathcal{P}_i$ under POED;

The first step is to allocate main tasks in Γ (line 2). Without the need to dedicate processors for backup tasks, all processors in the system are accessible to the main tasks. Again, we assume that the WFD heuristic is adopted to balance the workload of main tasks among the processors (line 3).

After that, the second step is to allocate the backup tasks in Γ^B to all processors (lines 4 and 5). Recall that, to tolerate a single permanent fault, a main task T_i and its backup task B_i have to be allocated to different processors [33]. Following this principle, we consider in this work two approaches when allocating backup tasks.

Cyclic backup allocation: First, considering that the WFD partition obtained in the first step has relatively balanced workload of main tasks among the processors, a simple approach is the *Cyclic Allocation* of the backup tasks. That is, for the main tasks allocated to processor $\mathcal{P}_i$, the corresponding backup tasks will be mapped to the next neighbor processor $\mathcal{P}_{i+1}$ and so on $(i = 1, \ldots, m-1)$. For the main tasks on the last processor $\mathcal{P}_m$, their backup tasks are allocated to the first processor $\mathcal{P}_1$, forming a cyclic chain allocation of backup tasks (and the scheme is denoted as *POED-Cyclic*).

The cyclic allocation is easy to implement and can simplify the communication among processors at runtime when no permanent fault occurs. Here, the backup task of a main task can always be found on its next neighbor processor and vice versa. However, once a processor fails, the recovery steps can be quite complicated to re-establish such a cyclic allocation of backup tasks, which may require all tasks to be re-mapped among the remaining processors and have a rather long recovery window.

Mixed backup allocation: To avoid such cyclic dependency between processors, the second approach is to *scatter* backup tasks among all processors. Specifically, by considering one processor $\mathcal{P}_i$ $(i = 1, \ldots, m)$ at a time, the corresponding backup tasks of its main tasks are allocated to all other processors. Again, for the purpose of load-balancing, the WFD mapping heuristic is adopted. At the end, each processor will be allocated a completely mixed set of main and backup tasks and thus the scheme is denoted as *POED-Mix*.

After backup tasks are allocated, each processor $\mathcal{P}_i$ will have a subset Γ_i of main tasks and a subset Γ_i^B of backup tasks. Suppose that, for every processor, its allocated main and backup tasks are schedulable under POED. That is, we have $U(\Gamma_i) + U(\Gamma_i^B) \leq 1$ $(i = 1, \ldots, m)$. As the third step, the spare capacity (i.e., *static slack*) in the amount of $(1 - U(\Gamma_i) - U(\Gamma_i^B))$ on each processor $\mathcal{P}_i$ is exploited and the scaled frequency for the main tasks on that processor is calculated accordingly (lines 7 and 8). Then, the scaled frequency and the maximum frequency are assigned to the main and backup tasks, respectively (lines 9 and 10).

As mentioned previously, to cancel as much execution of backup tasks as possible at runtime, they should be delayed to the maximum extent and are given the ALAP preference while the main tasks have the ASAP preference on each processor (lines 13 and 14). Moreover, the *inflated* system utilization on each processor, which takes the scaled frequencies of main tasks into consideration, is ensured to be no more than 1. Therefore, after frequency assignment for the (main and backup) tasks, they are guaranteed to be schedulable on each processor under POED (from Theorem 1). Hence, the last step is to execute the tasks on each processor under POED, which is actually the online phase of the POED-based schemes (line 15).

Since backup tasks run at the maximum frequency, the system reliability with respect to transient faults can be preserved (in the absence of permanent faults). Moreover, as both the POED-Cyclic and POED-Mix schemes schedule any main task and its backup task on different processors, they guarantee to tolerate a single permanent fault on any processor at runtime.

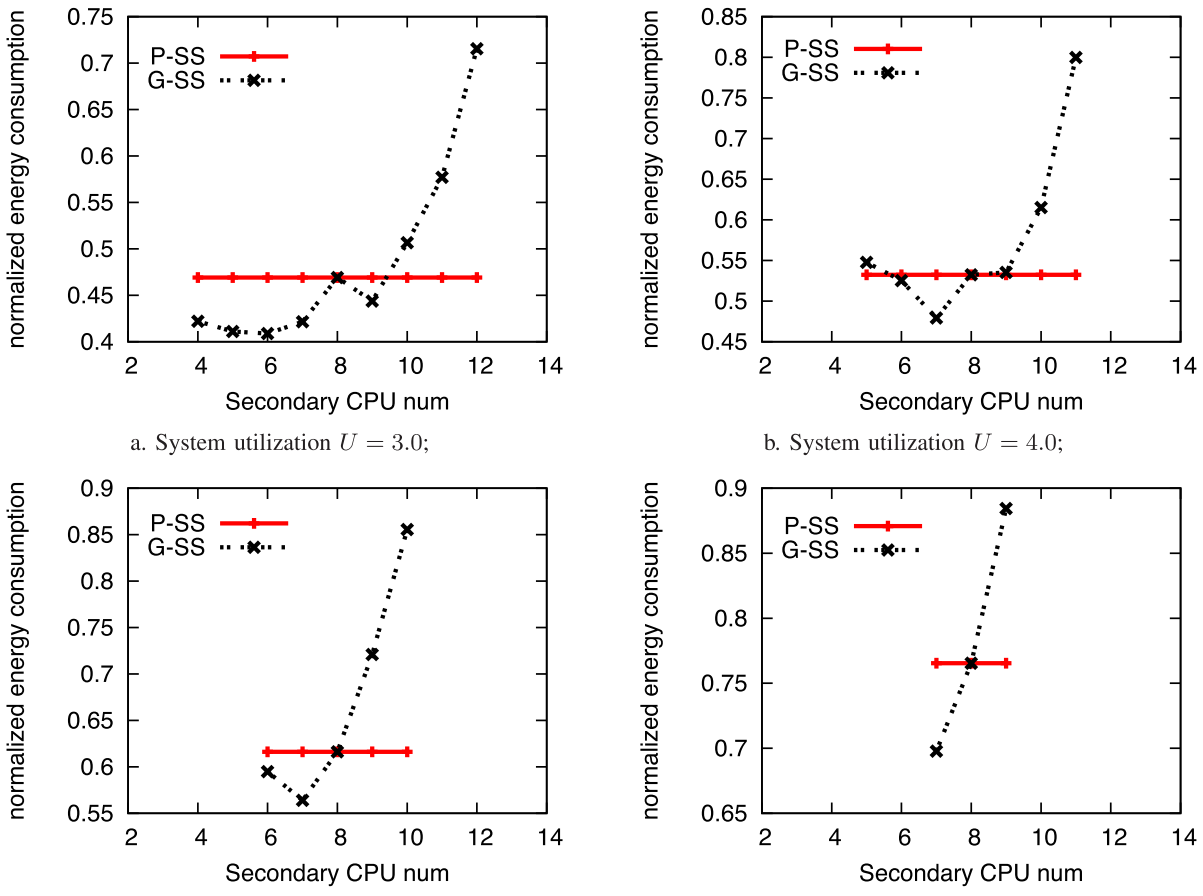
6. Evaluations and discussions

In this section, we evaluate the performance of the proposed SS-based and POED-based EEFT schemes for multiprocessor systems through extensive simulations. For such purposes, we developed a discrete event simulator using C++. From our previous studies [20,22], it has been shown that the Standby-Sparing and the POED-based schemes can preserve system reliability with respect to transient faults by enforcing backup tasks run at the maximum frequency in addition to the guarantees of tolerating a single permanent fault. Since the schemes studied in this paper follow the same design principle for fault tolerance, the reliability goals (in terms of tolerating both permanent and transient faults) can be ensured as well.

Therefore, in what follows, we focus on evaluating the energy efficiency of the proposed schemes only. Specifically, we show their normalized energy consumption, where the one under the *basic P-SS* with DPM only (i.e., both primary and secondary processors operate at the maximum frequency to execute tasks and sleep when idle) is used as the baseline.

Considering the fact that most modern processors have a few frequency levels [1,9], we assume that there are seven frequency levels, which are normalized as {0.4, 0.5, 0.6, 0, 7, 0.8, 0.9, 1.0} in the evaluations. Moreover, for the parameters in the power model, we assume that $P_{ind} = 0.01$, $C_{ef} = 1$ and $k = 3$, where similar parameters have been used in previous studies [22,52]. Moreover, we consider a system with up to 16 processors.

The utilizations of tasks are generated according to the *UUniFast* scheme proposed in [7], where the average task utilization is set as $u^{ave} = 0.1$ and $u^{ave} = 0.05$, respectively. For each task set, we generate enough number of tasks so that the system utilization reaches a given target value. That is, for a given system utilization U, the average number of tasks in a set will be $\frac{U}{u^{ave}}$. The periods of tasks are uniformly distributed in the range of [10, 100] and the



a. System utilization $U = 3.0$; b. System utilization $U = 4.0$;

c. System utilization $U = 5.0$; d. System utilization $U = 6.0$;

Fig. 3. The effects of XY-configuration in G-SS for a 16-CPU system under different loads; $u^{ave} = 0.1$.

WCET of a task is set according to its utilization and period. Each data point in the figures corresponds to the average result of 100 task sets.

6.1. Energy-Efficiency of G-SS vs. P-SS

First, we illustrate the variations in energy consumption under different configurations of primary and secondary processors in the G-SS scheme for a 16-processor system and compare them against that of the P-SS scheme. Here, we assume that all tasks run at their statically assigned frequencies and take their WCETs at runtime. Note that, due to independent scheduling of tasks' main and backup copies under EDF and EDL, respectively, it is possible for a task's backup copy finishes earlier than its main copy in the SS scheme [22]. Moreover, it is assumed that no fault occurs during the execution of tasks and backup (main) copies of tasks are cancelled under both schemes once their corresponding main (backup) copies complete successfully.

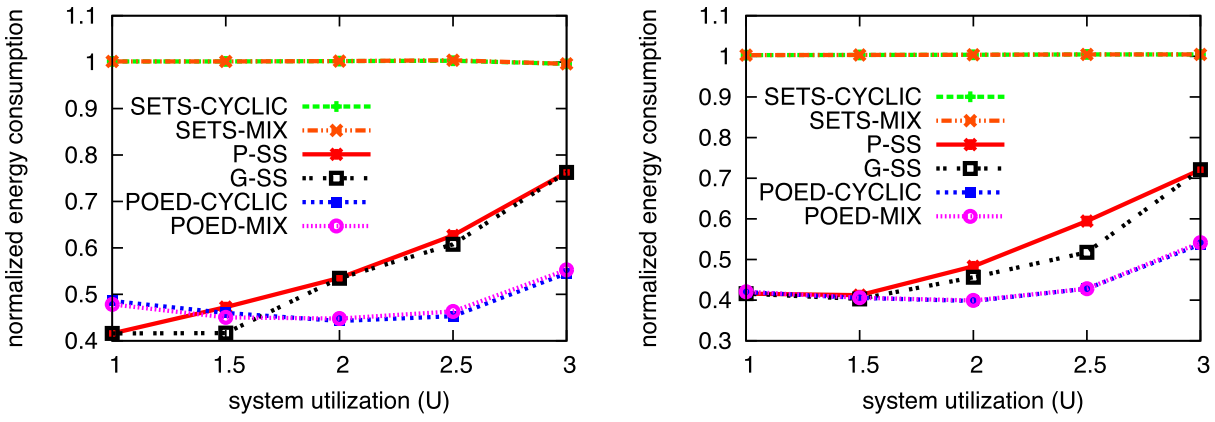
For a 16-processor system, the upper-bound of the total main task system utilization schedulable under the proposed schemes would be 8 (since the same processor capacity should be reserved for backup tasks). For the cases of system utilization $U = 3.0, 4.0, 5.0$ and 6.0, Fig. 3 shows the results for the G-SS scheme with varying numbers (Y) of secondary processors as well as that of the P-SS scheme for comparison. Here, the average task utilization is set as $u^{ave} = 0.1$.

Not surprisingly, for different processor configurations (i.e., as the number of secondary processors varies) in the G-SS scheme, the system energy efficiency can have rather large differences (from 30% to 45%). As illustrated in the example in Section 4.1, for a given system utilization, the configuration of processors that can lead to the best system energy efficiency normally has more primary processors (i.e., smaller values of Y). On the other hand, since the backup copies of tasks have to be executed at the maximum frequency for reliability preservation [22], the spare capacity on secondary processors is normally wasted, which leads to inferior performance for G-SS when more processors are used as secondaries.

From the results, we can also see that, with a judicious selection of the processor configurations (i.e., the values of X and Y), G-SS can outperform P-SS with up to 7% more energy savings. In the remaining evaluations, for any given task set, we assume that the G-SS scheme always adopts the most energy-efficient processor configuration.

6.2. Performance without slack reclamation

Next, we evaluate the schemes that do not consider online slack reclamation (which will be evaluated in the next section). In addition to the proposed SS- and POED-based schemes where the scaled frequencies are statically determined, for comparison, we implemented a *modified* SETS scheme [38], which considers only DPM and runs all tasks at the maximum frequency. SETS saves energy by delaying the executions of backup tasks through iteratively calculated latest scheduling times of those tasks in the EDF schedule [38]. Here, both cyclic and mixed mappings are considered, which are denoted as *SETS-Cyclic* and *SETS-Mix*, respectively.



a. $u^{ave} = 0.1$ b. $u^{ave} = 0.05$

Fig. 4. Performance of the offline schemes in a 8-CPU system.

The results are shown in Fig. 4, where tasks are assumed to take their WCETs and no fault occurs during the execution. Note that, SETS needs to consider all task instances within the LCM of tasks' periods. To obtain the results in reasonable time, we consider a 8-processor system and limit the LCM of task sets to be 12,000. Here, without considering the more effective DVFS technique, we can see that the SETS schemes perform almost the same as the basic P-SS scheme, which deploys only DPM as well. However, by exploiting the DVFS technique, the SS- and POED-based schemes can obtain significantly more energy savings than SETS schemes.

The results also show that, compared to P-SS, G-SS with the most energy-efficient configuration can always perform better under different system utilizations. However, the performance difference between P-SS and G-SS diminishes at very low or high system utilizations. The reason is that, at low system utilizations (i.e., $U \leq 1$), all main tasks can be executed at 0.4 (the lowest available frequency) while most backup tasks can be cancelled under both schemes. At high system utilizations (e.g., $U = 3.0$), there is only one feasible configuration (i.e., $X = 4$) for the G-SS scheme, which makes G-SS to act exactly the same as P-SS due to the same WFD heuristic adopted when partitioning main and backup copies of tasks.

For the POED-based schemes, POED-Cyclic and POED-Mix have very close performance on energy savings even though they have different backup partitions. However, in most cases, POED-based schemes can outperform P-SS and G-SS with up to 20% more energy savings. The reason is that, with mixed allocation of main and backup tasks on the processors, POED-based schemes can better utilize the available slack to slow down main tasks and reduce the overlapped execution with their corresponding backup tasks.

When $u^{ave} = 0.1$ (i.e., relatively large tasks), Fig. 4(a) shows that POED-based scheme may perform worse compared to that of the SS-based schemes when system utilization is very low ($U \leq 1.5$). This comes from the fewer number of available tasks, which cause unbalanced partitions of tasks among the processors under the POED-based schemes. For smaller tasks (i.e., $u^{ave} = 0.05$) where there are more tasks for the same system utilization, Fig. 4(b) shows that the POED-based schemes perform no worse than the SS-based schemes.

6.3. Performance with online slack reclamation

It is well-known that real-time tasks normally take only a small fraction of their WCETs at runtime [16]. In addition, most backup tasks will be cancelled at runtime as faults are rare events. Hence, significant amount of dynamic slack can be expected, which should be exploited to further scale down main tasks for more energy savings.

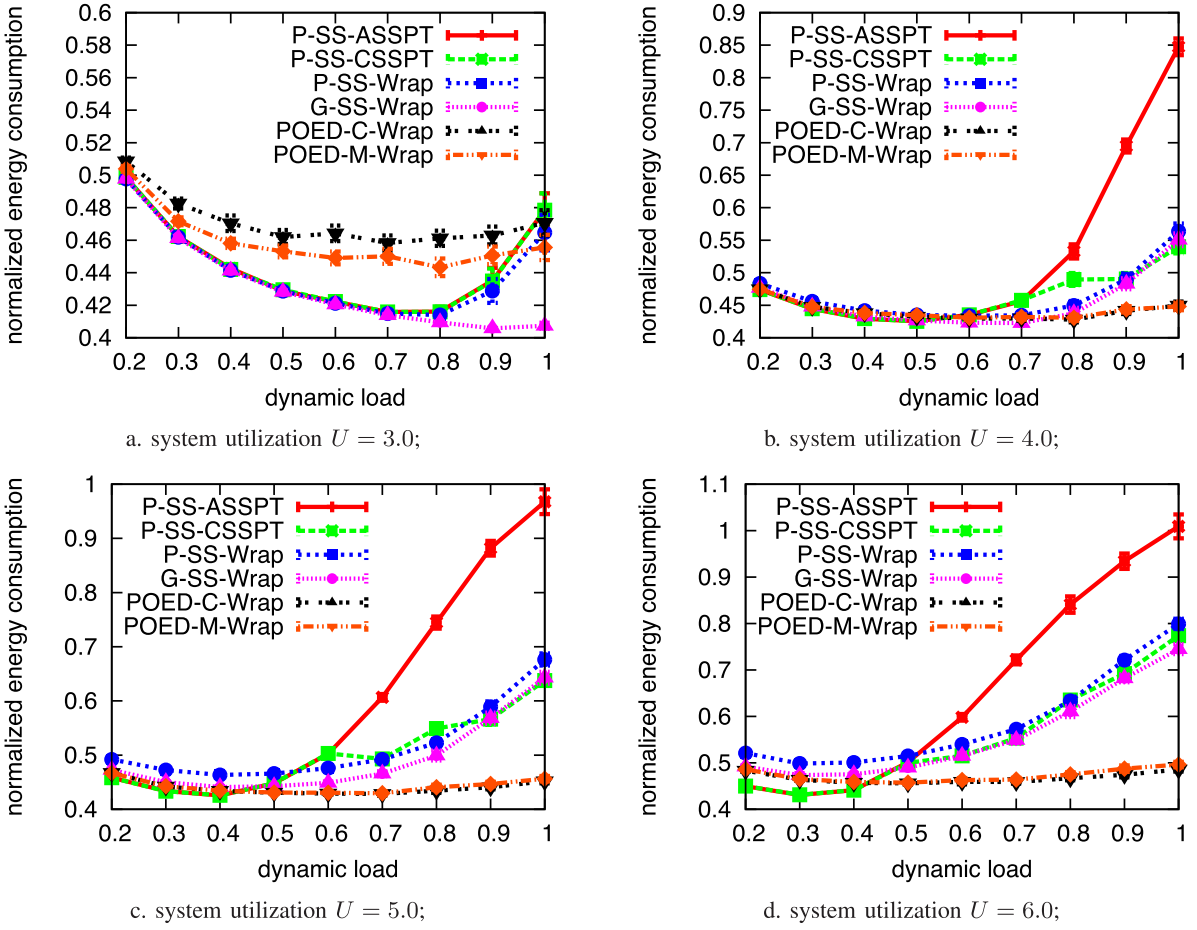
In this section, by varying the ratio of average over worst case execution times of tasks, we further evaluate the performance of the SS-based and POED-based schemes with an online power management scheme based on the wrapper-task technique [48]. For comparison, we also implemented both ASSPT and CSSPT techniques [22] for the P-SS scheme, which are denoted as "P-SS-ASSPT" and "P-SS-CSSPT", respectively. For the online scheme based on wrapper-tasks, it can be applied to the primary processors under both P-SS and G-SS, which are denoted as "P-SS-Wrap" and "G-SS-Wrap", respectively. The POED-based schemes enhanced with the online wrapper-tasks based technique are further denoted as "POED-C-Wrap" and "POED-M-Wrap", respectively.

Here, we consider 16 processors and set $u^{ave} = 0.1$. To emulate the dynamic execution behaviors of tasks, we use a system wide average-to-worst case execution time ratio α. For each task T_i, its average-to-worst case execution time ratio α_i is generated randomly around α. Then, at run-time, the actual execution time for each instance of task T_i is randomly generated around $\alpha_i \cdot c_i$, where c_i is task T_i's WCET. Essentially, α indicates the amount of dynamic slack that will be available at runtime where lower values indicate more slack.

Fig. 5 show the performance of the schemes with varying α (average-to-worst case execution times of tasks) under various utilizations (i.e., $U = 3.0, 4.0, 5.0$ and 6.0, respectively). Again, when the system utilization is low (i.e., $U = 3.0$), the main tasks can be executed at the lowest frequency of 0.4 and most backup tasks are cancelled, which leads to very close (within 6% difference) normalized energy consumption for P-SS and G-SS with different online techniques.

For cases with $\alpha = 1$, there is no dynamic slack at run-time. However, due to the limitation of discrete frequencies, there will be some spare capacity on each primary processor, which can be exploited by the wrapper-task based schemes and some additional energy savings can be obtained when compared to that of ASSPT and CSSPT. Therefore, with the limited benefits of online techniques with $\alpha = 1$, G-SS outperforms P-SS slightly, which is consistent with the results in the last section.

When the system utilization gets higher (i.e., $U = 4.0, 5.0$ and $U = 6.0$), we can see that the ASSPT technique can cause dramatic performance degradation for P-SS as the dynamic load of tasks increases (i.e., with higher values of α). The results are in line with what have been reported in [22]. The reason comes from the aggressive slack usage under the ASSPT technique, which executes the main tasks at very low frequency at the beginning of



a. system utilization $U = 3.0$; b. system utilization $U = 4.0$;

c. system utilization $U = 5.0$; d. system utilization $U = 6.0$;

Fig. 5. Performance of both SS-based and POED-based schemes with online slack reclamation under different system loads.

the schedule. Such scaled executions force remaining main tasks to run at much higher frequencies and cause more overlapped executions with their backup tasks on the secondary processors.

To address the above mentioned problem, based on the static and dynamic loads of tasks, the CSSPT scheme statically determines a lower bound for the scaled frequency for executing tasks' main copies when reclaiming slack at run-time [22]. With such a scaled frequency bound, CSSPT can effectively prevent the aggressive usage of slack time in the early part of the schedule. Therefore, when compared to ASSPT, P-SS performs much better with the CSSPT online technique, especially for tasks with higher dynamic loads.

For the wrapper-task based online technique, we can see that its performance is pretty stable under different dynamic loads of tasks. Although it performs (slightly) worse than that of ASSPT and CSSPT for the P-SS scheme at low dynamic loads (i.e., $\alpha \leq 0.5$), its performance is very close to that of CSSPT at higher dynamic loads of tasks. However, different from CSSPT, the wrapper-task based online technique does not requires the pre-knowledge of tasks' average-case workloads.

Moreover, as a generic online technique, wrapper-tasks can also be applied to the primary processors in the G-SS scheme, which is shown to have a stable performance as well. Although the performance gain of applying the wrapper-task technique on G-SS is rather limited (within 5%) when compared to that of P-SS, we can see that G-SS-Wrap always performs better than that of P-SS-CSSPT at higher dynamic loads of tasks.

With the POED-schemes, when the utilization is low, the main tasks can be executed at the lowest frequency 0.4 and most backup tasks can be cancelled. However, as before, due to the unbalanced workload distribution at very low system utilization (e.g., $U = 3.0$), POED-based schemes can have slightly inferior performance compared to SS-based schemes.

Moreover, as system utilization increases (i.e., for the cases of $U = 4.0, 5.0$ and $U = 6.0$), both POED-Cyclic and POED-Mix with wrapper-task based online technique can achieve much better and more stable energy savings comparing with the SS-based schemes. Again, this comes from the fact that with more workload in the system, both POED-based schemes can utilize the available system resource (CPU time) more efficiently. Specifically, by mixing the main and backup tasks on all processors, with the wrapper-task based online technique, all available (static and dynamic) slack time can be exploited to slow down the execution of main tasks and/or delay the execution of backup tasks, which results in much reduced overlapped executions (thus less energy consumption).

7. Conclusions

In this paper, we study energy-efficient fault-tolerance (EEFT) schemes for periodic tasks running on multiprocessor systems with the objectives of tolerating a single permanent fault while preserving system reliability with respect to transient faults. Based on the idea of Standby-Sparing (SS) technique, we first propose both *Paired-SS* and *Generalized-SS* schemes. Then, based on the preference-oriented earliest deadline (POED) scheduler, we study two POED-based schemes (i.e., *POED-Cyclic* and *POED-Mix*). The simulation results show that, for systems with a given number of processors, there normally exists a processor configuration where

the Generalized-SS scheme can have better energy savings compared to that of the Paired-SS scheme. Both SS- and POED-based schemes can obtain better energy savings compared the existing SETS scheme. Moreover, the POED-based schemes generally outperform the SS-based schemes in terms of energy savings, especially for systems with modest to high system loads.

References

[1] AMD, Amd opteron quad-core processors, 2009, http://www.amd.com/us/products/embedded/processors/.

[2] S. Aminzadeh, A. Ejlali, A comparative study of system-level energy management methods for fault-tolerant hard real-time systems, IEEE Trans. Comput. 60 (September(9)) (2011) 1288–1299.

[3] H. Aydin, V. Devadas, D. Zhu, System-level energy management for periodic real-time tasks, in: Proceedings of The 27th IEEE Real-Time Systems Symposium (RTSS), 2006, pp. 313–322.

[4] H. Aydin, Q. Yang, Energy-aware partitioning for multiprocessor real-time systems, Proceedings of the Parallel and Distributed Processing Symposium (IPDPS), 2003. April.

[5] A.A. Bertossi, L.V. Mancini, A. Menapace, Scheduling hard-real-time tasks with backup phasing delay, Proceedings of the IEEE International Symposium on Distributed Simulation and Real-Time Applications, 2006.

[6] A.A. Bertossi, L.V. Mancini, F. Rossini, Fault-tolerant rate-monotonic first-fit scheduling in hard-real-time systems, IEEE Trans. Parallel Distrib. Syst0 10 (9) (1999) 934–945.

[7] E. Bini, G.C. Buttazzo, Biasing effects in schedulability measures, Proceedings of the Euromicro Conference on Real-Time Systems, 2004.

[8] H. Chetto, M. Chetto, Some results of the earliest deadline scheduling algorithm, IEEE Trans. Softw. Eng. 15 (1989) 1261–1269.

[9] Intel Corp., Intel embedded quad-core xeon, 2009, http://www.intel.com/products/embedded/processors.htm.

[10] F. Dabiri, N. Amini, M. Rofouei, M. Sarrafzadeh, Reliability-aware optimization for DVS-enabled real-time embedded systems, in: Proceedings of the International Symposium on Quality Electronic Design, 2008, pp. 780–783.

[11] V. Degalahal, L. Li, V. Narayanan, M. Kandemir, M.J. Irwin, Soft errors issues in low-power caches, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 13 (October(10)) (2005) 1157–1166.

[12] A. Ejlali, B.M. Al-Hashimi, P. Eles, A standby-sparing technique with low energy-overhead for fault-tolerant hard real-time systems, in: Proceedings of the IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis, 2009, pp. 193–202.

[13] A. Ejlali, M.T. Schmitz, B.M. Al-Hashimi, S.G. Miremadi, P. Rosinger, Energy efficient seu-tolerance in DVS-enabled real-time systems through information redundancy, in: Proceedings of the Interantional Symposium on Low Power and Electronics and Design, 2005, pp. 281–286.

[14] E.M. Elnozahy, R. Melhem, D. Mossé, Energy-efficient duplex and TMR real–time systems, in: Procedings of The 23rd IEEE Real-Time Systems Symposium (RTSS), 2002, pp. 256–265.

[15] D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, N.S. Kim, K. Flautner, Razor: circuit-level correction of timing errors for low-power operation, IEEE Micro 24 (6) (2004) 10–20.

[16] R. Ernst, W. Ye, Embedded program timing analysis based on path clustering and architecture classification, in: Proceedings of The International Conference on Computer-Aided Design (ICCAD), 1997, pp. 598–604.

[17] S. Ghosh, R. Melhem, D. Mossé, Fault-tolerance through scheduling of aperiodic tasks in hard real-time multiprocessor systems, Parallel Distrib. Syst. IEEE Trans. 8 (March(3)) (1997) 272–284.

[18] Y. Guo, H. Su, D. Zhu, H. Aydin, Preference-oriented real-time scheduling and its application in fault-tolerant systems, J. Syst. Archit. 61 (2) (2015) 127–139.

[19] Y. Guo, D. Zhu, H. Aydin, Reliability-aware power management for parallel real-time applications with precedence constraints, in: Proceedings of the International Green Computing Conference (IGCC), 2011, pp. 1–8. July

[20] Y. Guo, D. Zhu, H. Aydin, Efficient power management schemes for dual-processor fault-tolerant systems, Proceedings of the First Workshop on Highly-Reliable Power-Efficient Embedded Designs (HARSH), in conjunction with HPCA, 2013. Feb.

[21] Y. Guo, D. Zhu, H. Aydin, Generalized standby-sparing techniques for energy–efficient fault tolerance in multiprocessor real-time systems, in: Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2013. August

[22] M.A. Haque, H. Aydin, D. Zhu, Energy-aware standby-sparing technique for periodic real-time applications, in: Proceedings of the IEEE International Conference on Computer Design (ICCD), 2011.

[23] M.A. Haque, H. Aydin, D. Zhu, Energy management of standby-sparing systems for fixed-priority real-time workloads, in: Proceedings Of the Second International Green Computing Conference (IGCC), 2013. June

[24] International technology roadmap for semiconductors, 2008, S. R. Corporation. http://public.itrs.net.

[25] S. Irani, S. Shukla, R. Gupta, Algorithms for power savings, in: Proceedings of the 14th Annual ACM-SIAM symposium on Discrete algorithms (SODA), 2003, pp. 37–46.

[26] R.K. Iyer, D.J. Rossetti, M.C. Hsueh, Measurement and modeling of computer reliability as affected by system activity, ACM Trans. Comput. Syst. 4 (3) (1986) 214–237. August

[27] V. Izosimov, P. Pop, P. Eles, Z. Peng, Design optimization of time-and cost-constrained fault-tolerant distributed embedded systems, in: Proceedings of Design, Automation and Test in Europe, 2005, pp. 864–869.

[28] N.-S. Kim, T. Austin, D. Blaauw, T. Mudge, K. Flautner, J.-S. Hu, M.-J. Irwin, M. Kandemir, V. Narayanan, Leakage current: Moore's law meets static power, Computer 36 (December(12)) (2003) 68–75.

[29] Z. Li, L. Wang, S. Li, S. Ren, G. Quan, Reliability guaranteed energy-aware frame-based task set execution strategy for hard real-time systems, J. Softw. Syst. (2013).

[30] W. Liao, L. He, K.M. Lepak, Temperature and supply voltage aware performance and power modeling at microarchitecture level, IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 24 (7) (2005) 1042–1053.

[31] R. Melhem, D. Mossé, E.M. Elnozahy, The interplay of power management and fault recovery in real-time systems, IEEE Trans. Comput. 53 (2) (2004) 217–231.

[32] P. Pop, K.H. Poulsen, V. Izosimov, P. Eles, Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems, Proceedings of the IEEE/ACM Interantional Conference on Hardware/software codesign and System Synthesis, 2007.

[33] D.K. Pradhan (Ed.), Fault-tolerant Computer System Sesign, Prentice-Hall, Inc, Upper Saddle River, NJ, USA, 1996.

[34] X. Qi, D. Zhu, H. Aydin, Global scheduling based reliability-aware power management for multiprocessor real-time systems, Real-Time Syst. 47 (2) (2011) 109–142.

[35] M.T. Schmitz, B.M. Al-Hashimi, P. Eles, System-Level Design Techniques for Energy-Efficient Embedded Systems, Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[36] R. Sridharan, N. Gupta, R. Mahapatra, Feedback-controlled reliability-aware power management for real-time embedded systems, in: Proceedings of the Design Automation Conference, 2008, pp. 185–190.

[37] M.-K. Tavana, M. Salehi, A. Ejlali, Feedback-based energy management in a standby-sparing scheme for hard real-time systems, Proceedings of the IEEE Real-Time Systems Symposium, 2011.

[38] O.S. Unsal, I. Koren, C.M. Krishna, Towards energy-aware software-based fault tolerance in real-time systems, in: Proceedings of the Interantional Symposium on Low Power Electronics and Design, 2002, pp. 124–129.

[39] T. Wei, P. Mishra, K. Wu, H. Liang, Online task-scheduling for fault-tolerant low-energy real-time systems, in: Proceedings of IEEE/ACM International Conference on Computer-Aided Design, 2006, pp. 522–527.

[40] T. Wei, P. Mishra, K. Wu, H. Liang, Fixed-priority allocation and scheduling for energy-efficient fault tolerance in hard real-time multiprocessor systems, IEEE Trans. Parallel Distrib.Syst. (TPDS) 19 (2008) 1511–1526.

[41] Y. Xiang, S. Pasricha, Soft and hard reliability-aware scheduling for multi-core embedded systems with energy harvesting, IEEE Trans. Multi-Scale Comput.Syst. 1 (October(4)) (2015) 220–235.

[42] Y. Zhang, K. Chakrabarty, Energy-aware adaptive checkpointing in embedded real-time systems, in: Proceedings of Design, Automation and Test in Europe (DATE), 2003, pp. 918–923.

[43] Y. Zhang, K. Chakrabarty, Task feasibility analysis and dynamic voltage scaling in fault-tolerant real-time embedded systems, in: Proceedings of Design, Automation and Test in Europe Conference(DATE), 2004, pp. 1170–1175.

[44] Y. Zhang, K. Chakrabarty, V. Swaminathan, Energy-aware fault tolerance in fixed-priority real-time embedded systems, in: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2003.

[45] B. Zhao, H. Aydin, D. Zhu, Energy management under general task-level reliability constraints, in: Proceedings of the 18th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), 2012, pp. 285–294. Apr.

[46] B. Zhao, H. Aydin, D. Zhu, Shared recovery for energy efficiency and reliability enhancements in real-time applications with precedence constraints, ACM Trans. Design Autom. Electron. Syst. (TODAES) 18 (2–23) (2013).

[47] D. Zhu, Reliability-aware dynamic energy management in dependable embedded real-time systems, ACM Trans. Embedded Comput. Syst. 10 (2) (2010) 26.1–26.27.

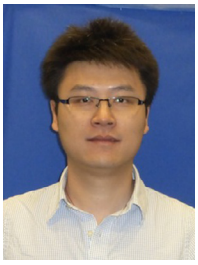
[48] D. Zhu, H. Aydin, Reliability-aware energy management for periodic real-time tasks, IEEE Trans. Comput. 58 (10) (2009) 1382–1397.

[49] D. Zhu, R. Melhem, D. Mossé, The effects of energy management on reliability in real-time embedded systems, in: Proceedings of the International Conference on Computer Aidded Design, 2004, pp. 35–40.

[50] D. Zhu, R. Melhem, D. Mossé, E.M. Elnozahy, Analysis of an energy efficient optimistic tmr scheme, in: Proceedings of the 10th Interantional Conference on Parallel and Distributed Systems, 2004.

[51] D. Zhu, D. Mossé, R. Melhem, Energy efficient redundant configurations for real-time parallel reliable servers, J. Real-Time Syst. 41 (April(3)) (2009) 195–221.

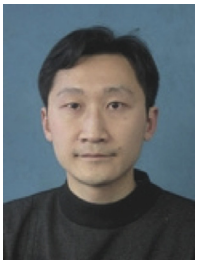
[52] S. Zhuravlev, J.-C. Saez, S. Blagodurov, A. Fedorova, M. Prieto, Survey of energy-cognizant scheduling techniques, IEEE Trans. Parallel Distrib. Syst. 24 (7) (2013) 1447–1464.



Yifeng Guo obtained the Ph.D. degree from the Department of Computer Science at the University of Texas at San Antonio in 2013. His research interests include real-time systems, multiprocessor and parallel systems, power management and fault tolerance.

Dakai Zhu received the Ph.D. degree in Computer Science from the University of Pittsburgh in 2004. He is currently a Professor in the Department of Computer Science at the University of Texas at San Antonio. His research is in the general area of real-time systems. He has served on technical program committees for several major real-time conferences (e.g., RTSS, RTAS and RTCSA). He was a recipient of the US National Science Foundation (NSF) Faculty Early Career Development (CAREER) Award in 2010. He is a member of the ACM and IEEE.

Hakan Aydin received the Ph.D. degree in computer science from the University of Pittsburgh in 2001. He is currently an associate professor in the Computer Science Department at George Mason University. He was a recipient of the US National Science Foundation (NSF) Faculty Early Career Development (CAREER) Award in 2006. His research interests include real-time systems, low-power computing, and fault tolerance. He is a member of the IEEE.



Jian-Jun Han received the Ph.D. degree in computer science and engineering from Huazhong University of Science and Technology (HUST) in 2005. He is now an Associate Professor at the School of Computer Science and Technology in HUST. He worked at the University of California, Irvine as a visiting scholar between 2008 and 2009, and at the Seoul National University between 2009 and 2010. His research interests include real-time system and parallel computing. He is currently a member of IEEE and ACM.

Laurence T. Yang is with the Department of Computer Science, St. Francis Xavier University, Antigonish, NS, Canada and holds adjunct position at Huazhong University of Science and Technology, China. His research interests include high performance computing and networking, embedded systems, ubiquitous/pervasive computing, and intelligence. His research is supported by National Sciences and Engineering Research Council, Canada and Canada Foundation for Innovation.