The CSI Journal on Computer Science & Engineering

The CSI Journal on
Computer Science and Engineering
Vol. 15, No. 1, 2017
Pages 31-38
Regular Paper

AdDQ: Low-Energy Hardware Replication for Real-Time Systems through Adaptive Dual-Queue Scheduling

Mohsen Ansari[1], Sepideh Safari[1], Farimah R.Poursafaei[1], Mohammad Salehi[2], and Alireza Ejlali[1]

[1]Department of Computer Engineering, Sharif University of Technology, Tehran, Iran
[2]Computer Engineering Department, University of Guilan, Rasht, Iran

Abstract

Low energy consumption and high reliability are two major design objectives for real-time embedded systems. Beside other techniques, hardware replication (e.g. standby-sparing) can provide high reliability while keeping the energy consumption under control. In this paper, we consider two replicated processors as a standby-sparing system where main copy tasks on primary are scheduled by Earliest-Deadline-First (EDF) while backup copy tasks on spare are scheduled by our proposed Adaptive Dual-Queue (AdDQ) scheduling. AdDQ provides the best opportunity to postpone the spare executions as much as possible to minimize execution overlaps between main and backup copy tasks. Therefore, when a copy task finishes successfully a larger portion of its corresponding copy task can be cancelled, resulting in a significant amount of energy saving. To achieve further energy saving, we use Dynamic Voltage Scaling (DVS) and, Dynamic Power Management (DPM). The main reason of using DPM is that, once a copy of task has finished successfully, its other copy task is terminated, and if there is no more task for execution the processors go to a low-power mode. We evaluated our AdDQ technique under various system configurations. Experiments show that AdDQ provides up to 36% (on average by 14%) energy savings compared to four state-of-the-art techniques.

Keywords: Real-time Embedded System, Energy Management, Hardware Replication, Scheduling.

1. Introduction and Related Work

Energy consumption is an important design concern for real-time embedded systems [1, 2]. One of the well-known techniques to manage the energy is Dynamic Voltage Scaling (DVS) which reduces energy consumption by scaling the processor supply voltage and operating frequency [3, 4, 5]. However, the applicability of DVS is limited by the amount of available slack time in the system [6, 7]. Another technique is Dynamic Power Management (DPM), where the system components are put into sleep mode when they are temporarily unused [4, 8]. The other important design concerns in designing real-time embedded systems are high reliability and fault tolerance [6, 9, 10, 11]. Faults in computer systems are classified into transient, intermittent and permanent [10]. Transient faults are often induced by electromagnetic interference and cosmic radiations [10, 12, 13]. Transient faults in real-time embedded systems can be tolerated using time redundancy (e.g. re-execution and roll-back recovery [10, 12, 14]). Also, permanent hardware faults result from hardware component failure or manufacturing defects. They are usually mitigated through exploiting replicated hardware component [10, 12]. However, hardware replication techniques may incur significant energy over-head to the system [6], [9]. Therefore, there is a need to manage the energy consumption overhead of fault tolerance techniques that are exploited for real-time embedded systems .

Some techniques, like [14, 15, 16, 17, 27], which consider both reliability and energy consumption, reserve a part of the available slack time to schedule a recovery task (to preserve

the system reliability), and then utilize the remaining slack for energy savings. In these techniques, since both the main and recovery tasks are executed on the same processor, tasks with utilization greater than 50% cannot be scheduled. Furthermore, these techniques cannot tolerate permanent faults, since both the main and recovery executions perform on the same processor. Standby sparing [6, 18, 19, 20] is a well-studied hardware replication technique to provide high reliability while keeping the energy consumption under control. In standby sparing, the system consists of two identical processors: primary and spare. Main tasks are executed on primary and their backup tasks are executed on spare. When the primary processor fails (due to either transient or permanent fault), it is replaced with the spare processor to continue the execution of the backup task .

To reduce the energy consumption overhead of standby sparing, [2] has proposed a technique where DVS is used for the primary processor while the spare processor does not use DVS to preserve the reliability of the system when a fault occurs. Upon the successful completion of a main task, the corresponding backup task is cancelled and excessive energy consumption is avoided. The scheme proposed in this work is suitable for non-preemptive and aperiodic tasks. While, most of the real-time applications on embedded systems are inherently periodic. The work in [19] has proposed an energy-aware scheduling scheme for a standby-sparing system that executes preemptive periodic real-time applications. They apply Earliest-Deadline-First (EDF) scheduling with DVS on the primary processor, while the backup tasks are executed on the spare processor according to Earliest-Deadline-Late (EDL) scheduling. Both EDF and EDL assign priorities based on the jobs deadline, however EDL delays the jobs as much as possible to obtain idle intervals as early as possible in the schedule. They aim at minimizing the execution overlap between main and backup tasks at run-time to reduce energy consumption. They have presented two algorithms denoted as Aggressive Standby-Sparing for Periodic Tasks (ASSPT) and Conservative Standby-Sparing for Periodic Tasks (CSSPT). They differ in the way that they use the available slack at run-time for frequency assignment. ASSPT allows a task to aggressively utilize the entire available slack time to reduce operational frequency as much as possible. CSSPT aims at achieving a balanced slack time distribution among all tasks. Since ASSPT and CSSPT use the slack time on spare to apply DVS on primary, in some conditions primary tasks may miss their deadlines. Also, these schemes do not use DPM on the primary and spare processors, while putting a processor into the sleep/low-power mode when it is idle can pro-vide further energy saving. [20] has proposed an energy-management technique for a standby sparing system that executes preemptive fixed-priority real-time tasks. Tasks on the primary processor are scheduled by the Cycle-Conserving DVS algorithm that has been proposed for Rate Monotonic Scheduling (RMS) in [5]. While the spare uses DPM and dual-queue mechanism that tries to maximally delaying the backup tasks to save more energy. It should be noted that, although RMS is optimal for fixed priority tasks, it lowers processor utilization.

1.1. Concept Overview and Our Novel Contribution

In this paper, we consider a dual-processor standby-sparing system that executes preemptive periodic real-time tasks. We apply the Earliest-Deadline-First (EDF) scheduling on the primary processor with DVS and DPM. For the spare processor we propose an adaptive dual-queue scheduling. Dual-queue scheduling postpones the execution of backup tasks, as much as possible [20]. Furthermore, we introduce an approach to apply DPM on the spare processor to achieve more energy saving .

Adaptive Dual-Queue (AdDQ) Scheduling: The idea of dual-priority scheduling was proposed in [22]. It is a strategy for scheduling periodic, sporadic and adaptive tasks with both soft and hard deadlines in real-time systems. It is assumed that the system has three queues according to the tasks execution priorities: Upper, Middle and Lower. Under dual-priority scheduling, hard tasks (i.e. tasks with hard deadlines) are executed either on an upper or lower priority queue. At run-time, when a hard task releases, it is put into the lower queue. However, after a while, the task is promoted to the upper queue. Other tasks, typically with firm or soft deadlines, are put into the middle queue. The main challenge of dual-priority scheduling is to determine the promotion time for hard tasks, to make sure that they will eventually meet their deadlines in the upper queue.

Zhu *et al*. [20] have proposed a dual-queue mechanism based on dual-priority scheduling, which is applied to the spare processor. Their mechanism has two queues denoted as lower and upper. When a job arrives, it is put into the lower queue and after the corresponding promotion time it is promoted to the upper queue for execution. The promotion time is computed statically for each task at design time. The mechanism in [20] schedules fixed-priority tasks and jobs in the upper queue based on RMS.

Our AdDQ technique is based on [20] and the only difference is that in AdDQ, jobs in the upper queue are scheduled by EDF while in [20] they are scheduled by RMS. It adaptively updates the promotion time to more delay backup executions at run-time according to available slack time. The slack time releases when a primary task finishes successfully and its corresponding backup execution is canceled (Section 3) .

The rest of the paper is organized as follows. In section 2 we present task, energy and fault models. In section 3, we present the details of our solution. Our experimental results are shown in section 4 and we conclude the paper in section 5.

2. Model and Assumptions

2.1. Application Model

We consider a set of periodic real-time tasks $\psi=\{\tau_1,\ldots, \tau_n\}$. Each task τ_i has a period P_i, a worst-case execution time WC_i (under the maximum frequency), and an actual execution time AC_i. The j-th job of a task τ_i ($J_{i,j}$) arrives at time $r_{i,j}=(j-1)\times P_i$ and must complete by its deadline $j\times P_i$. Hence, the relative deadline D_i of the job $J_{i,j}$ is equal to the period P_i. The utilization of the task τ_i is defined as WC_i/P_i. So, the sum of all tasks utilization is U_{tot}. We consider for each task τ_i a

backup task B_i. τ_i and B_i have the same timing parameters (i.e. P_i and WC_i). We denote the j-th job of B_i by $B_{i,j}$.

2.2. Energy Consumption Model

Each processor can operate in active and sleep modes. The processor executes tasks in the active mode and in this mode we compute its energy consumption based on Eq. 1. Total energy consumption of the system consists of static and dynamic energy components [18]. The static energy (E_s) is dominated by the leakage current. Dynamic energy (E_d) is mainly consumed due to system activity.

$$E_{total} = E_s + E_d \quad (1)$$

Under DVS, the voltage V_i that is used for the execution of the task τ_i may be less than the maximum voltage V_{max}. We denote the normalized voltage ρ_i as:

$$\rho_i = \frac{V_i}{V_{\max}} \quad (2)$$

Hence, the dynamic energy consumption under the scaled voltage V_i can be written as:

$$E_d(\tau_i) = C_{eff} V_i^2 f_i \left(\frac{AC_i}{\rho_i}\right) \quad (3)$$

where, C_{eff} is the average switched capacitance, V_i and f_i are supply voltage and operational frequency that is used to execute each task τ_i, and AC_i/ρ_i is the scaled task execution time under ρ_i. Let V_{max} be the maximum voltage corresponding to the maximum frequency f_{max}. Considering the almost linear relationship between voltage and frequency [18], we can write: $\rho_i = V_i/V_{max} = f_i/f_{max}$. Therefore, Eq. 3 can be written as:

$$E_d(\tau_i) = C_{eff} V_{\max}^2 f_{\max} \rho_i^2 AC_i \quad (4)$$

Since $C_{eff} V_{max}^2 f_{max}$ is constant, the energy consumption can be normalized by removing $C_{eff} V_{max}^2 f_{max}$. Therefore, the normalized energy consumption of the processor while executing the task τ_i can be written as:

$$NE_d(\tau_i) = \rho_i^2 AC_i \quad (5)$$

In this paper, the static energy E_s is set to 15% of the maximum dynamic energy, like the works [19], [20]. In order to apply DPM on both processors, let assume we have a *break to sleep time* ($\Delta_{critical}$). When the idle time of a processor is greater than $\Delta_{critical}$, processor switches to sleep mode, and hence, all energy components other than the static energy E_s are removed.

2.3. Fault Model

We consider a transient fault model similar to [19], [20]. The average fault rate λ is dependent on the processor frequency where by decreasing processor frequency, λ increases exponentially. The average fault rate on the frequency f can be expressed as:

$$\lambda(f) = \lambda_0 \times 10^{\frac{d(1-f)}{1-f_{\min}}} \quad (6)$$

where $\lambda_0 = 10^{-7}$ is the transient fault rate at f_{max} and d determines the sensitivity of the system to voltage scaling. Like the works [19], [20], we consider d=2 in this paper.

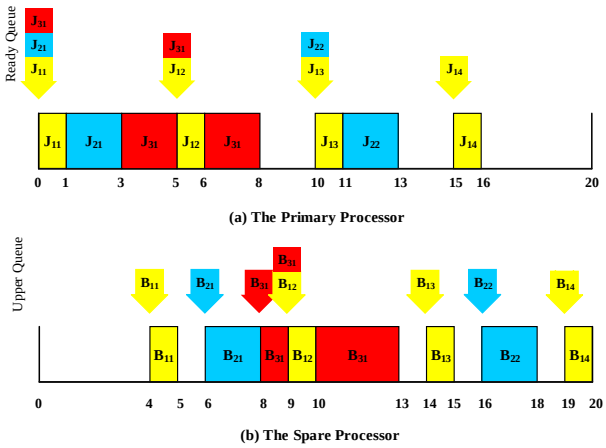

Fig. 1. Scheduling the example task set in Section 3. A through: (a) EDF on primary, (b) our AdDQ-EDF on spare. In this figure, DVS is not used and the tasks are executed on the maximum frequency.

3. Our Proposed Adaptive Dual-Queue Technique

3.1. Illustrative Example

Our AdDQ technique executes preemptive periodic real-time tasks on a standby-sparing system. On the primary processor, we use EDF scheduling with DVS and DPM. The spare processor uses adaptive dual-queue scheduling with DPM. As an example let us consider three periodic task τ_1, τ_2 and τ_3 with the period P_i and worst-case execution time WC_i: P_1=5, WC_1=1, P_2 =10, WC_2 =2, P_3 =20, WC_3=4. For this task set, the total utilization is 0.6 and the hyperperiod is H=20 (which is the least common multiple of all the task periods). Therefore, within a hyperperiod, 4 jobs of τ_1, 2 jobs of τ_2 and 1 job of τ_3 are executed. Fig. 1 shows how this task set is scheduled and executed by AdDQ. In this example, for ease of presentation, we temporarily do not exploit DVS. On the primary processor, the tasks are scheduled by the use of the preemptive Earliest-Deadline-First (EDF) scheduling (Fig. 1a). Note that if the main and backup tasks are scheduled in the same way on the primary and spare processors (e.g. both are scheduled with EDF), the energy consumption will significantly increases. This is because main tasks are executed with their backups in parallel. The energy overhead can be reduced through delaying the execution of backup tasks on the spare processor [2]. However, the deadlines of the backup tasks have to be guaranteed. To address this issue, for the spare processor (Fig. 1b), we exploit the adaptive dual-queue scheduling which is based on the dual-queue mechanism presented in [20]. In dual-queue scheduling we have two queues that we call them lower and upper queues. When a job arrives, it is put into the lower queue. In order to promote jobs to upper queue for execution we should find the exact promotion time, since tasks should not miss their deadlines in the upper queue. The first step in finding the exact promotion time is computing the worst-case response time of each task. There are multiple techniques to compute the worst-case response time of the tasks, e.g. [23]. We propose Eq. 7 for dynamic-priority tasks.

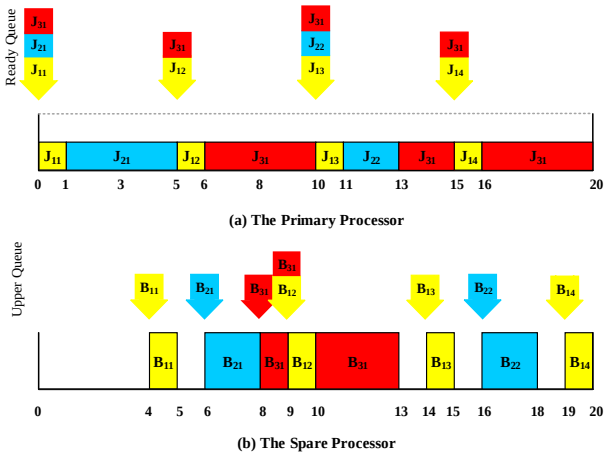

Fig. 2. Scheduling the example task set in Section III.A through: (a) EDF with DVS on primary, (b) our AdDQ-EDF on spare. Note that in our system DVS is only used for the primary processor.

$$S_i = \sum_{\tau_j \in hp(\tau_i)} \left\lfloor \frac{P_i}{P_j} \right\rfloor \times WC_j + WC_i \quad (7)$$

where $hp(\tau_i)$ is the set of tasks with priority higher than τ_i. The second step is computing the promotion time for τ_i by substituting Eq. 7 in Eq. 8 as:

$$Y_i = D_i - S_i \quad (8)$$

In Eq. 8, D_i is the deadline of τ_i and Y_i is its promotion time. Now we give an example to illustrate how AdDQ works. For the example task set, the promotion times ($Y_i = D_i - S_i$) can be computed as: $Y_1 = 4$, $Y_2 = 6$, and $Y_3 = 8$. Fig. 1b shows the delayed execution scenario. Despite the explicitly enforced delays, all the jobs still meet their deadlines.

Now, we consider a standby-sparing system where the DVS-enabled primary processor executes main tasks according to the EDF scheduling. Since the total utilization of the task set is 0.6, each task τ_i takes up to WC_i/f_i time units when executed at frequency f=0.6. The spare processor executes the backup jobs $\{Bi,j\}$ through the described adaptive dual-queue scheduling at the maximum frequency. Fig. 2 shows the corresponding schedules for the primary and spare processors. By applying dual-queue scheduling, the backup tasks on the spare processor are sufficiently delayed and their overlaps with the main tasks on the primary processor are reduced significantly.

By the use of the adaptive dual-queue scheduling further energy saving can be gained. Since, when a main job completes successfully or early, its corresponding backup job on the spare processor can be cancelled. Fig. 3 shows this fault free situation. For instance, assuming that J_{11}, J_{21} and J_{12} complete successfully on the primary, B_{11}, B_{21} and B_{12} are completely cancelled. J_{31} will be preempted by J_{13}. Note that, based on our AdDQ scheme B_{31} promotes to the upper queue at the time 8 (see Fig. 2b). However, in the fault-free state, B_{12} is canceled at the time 6 since J_{12} finishes successfully at that time, and hence, the promotion time of B_{31} is updated to 9 in Fig. 3b. When the backup copy B_{31} finishes at the time 13, we cancel the remaining parts of the main job J_{31} that are scheduled in the time slots 13-15 and 16-20 on the primary processor (Fig. 3a). Assuming that all the remaining main tasks complete successfully, we obtain the schedules in Fig. 3.

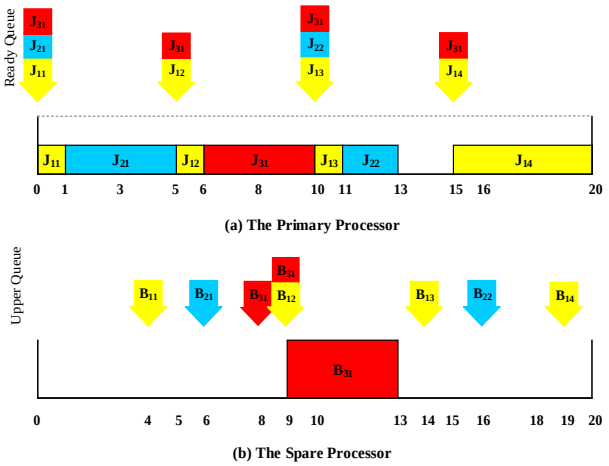

Fig. 3. Fault-free execution of Fig. 2.

3.2. Algorithm Discussion

In this sub-section, we present the details of our AdDQ technique. Main tasks are executed on the primary processor according to the EDF scheduling and based on the amount of the static and dynamic slack time, we apply DVS. The spare processor is reserved for the execution of backup tasks based on our adaptive dual-queue scheduling. Also, when the idle time of each of the processors is greater than the break to sleep time, DPM is applied. Algorithm 1 shows the event on the primary processor and the corresponding actions.

In Algorithm 1, in lines 1-15 when a job arrives, the main job is added to the ready queue (which is the only queue for primary processor), and its corresponding backup job is added to the lower queue on the spare. At run time, when a job is released, if the primary processor is idle, the job is executed. However, if the processor is running another job, the execution priorities are assigned according to EDF. If preemption occurs we update the minimum additional time required to complete the job ($WC_{m,n}$) in the worst-case (under the maximum frequency). In lines 16-20, we find slack time between t, i.e. the time that a job is executed or resumed from preemption, and its deadline. It should be noted that if there is no slack time, the job runs with the maximum frequency on the primary. To find the minimum frequency for a job execution, we use [19]:

$$f_{i,j} = \max(f_{ee}, U_{avg}, \frac{WC_{i,j}}{WC_{i,j} + slack}) \quad (9)$$

where f_{ee} is called the *energy-efficient speed* which is a processing frequency that below it, the total energy consumption of a task increases. f_{ee} can be computed analytically in advance [11] [24]. U_{avg} is the *average-case total utilization* of the task set. In this scheme, a job is not allowed to run at a frequency lower than U_{avg}. In lines 21-22, when a job completes on the primary, we call an acceptance test [10] to check the correctness of the task execution. In lines 23-25 if the acceptance test does not detect any fault, we cancel the corresponding backup task in the spare processor. Then, the primary processor continues with executing the next job in the ready queue. On the other hand,

Algorithm 1: Scheduler and Energy Manager for the Primary Processor

1. **Event – A job of τ_i (namely, $J_{i,j}$) is released at time t:**
2. Add $J_{i,j}$ to the ready queue on the primary
3. Add $B_{i,j}$ to the lower queue on spare
4. **If** the processor is busy **then**
5. **If** priority($J_{i,j}$) > priority($J_{m,n}$) **then**
6. **/*** preemption case */
7. $WC_{m,n} \leftarrow WC_{m,n} - f_{m,n} * e_{m,n}$
8. /* execution time of $J_{m,n}$ until now */
9. Execute($J_{i,j}$);
10. **else**
11. /* Execute $J_{m,n}$ */;
12. **end If**
13. **else /*** processor is idle */
14. Execute($J_{i,j}$);
15. **end If**
16. **Function** Execute($J_{i,j}$);
17. $slack \leftarrow$ EDF_Slack(t, deadline($J_{i,j}$))
18. $f_{i,j} \leftarrow \max(f_{ee}, U_{avg}, \frac{WC_{i,j}}{WC_{i,j}+slack})$
19. Execute $J_{i,j}$ on the primary processor at frequency $f_{i,j}$
20. **End Function**
21. **Event –** $J_{i,j}$ completes at time t:
22. Run the acceptance test
23. **If** no error is detected **then**
24. Cancel the backup $B_{i,j}$ on the spare processor
25. **end if**
26. **If** ready queue of primary is empty **then**
27. $earliest\_release \leftarrow$ time to earliest release time
28. $\Delta_{er} \leftarrow earliest\_release$
29. **If** $\Delta_{er} \geq \Delta_{critical}$ **then**
30. Put primary to sleep mode for Δ_{er} units of time
31. **end If**
32. **else** /* jobs are available for execution */
33. Execute a job on the primary
34. **end If**

Algorithm 2: Our AdDQ Scheduler and Energy Manager for the Spare Processor

1. **Event – Promotion of backup job $B_{i,j}$:**
2. Put $B_{i,j}$ into the upper queue on the spare
3. Execute the highest EDF priority job on spare
4. **Event –** Completion of backup job $B_{i,j}$:
5. Run the acceptance test for $B_{i,j}$
6. **if** $J_{i,j}$ is not completed yet **then**
7. Cancel $J_{i,j}$ on the primary
8. **end if**
9. **Event –** Cancellation of backup job $B_{i,j}$:
10. $S \leftarrow$ Slack time obtain from canceling the $B_{i,j}$
11. **for** every $B_{m,n}$ before the $B_{i,j}$ **then**
12. **if** $Y_{m,n}+S+WC_{i,j} \leq$ Deadline($B_{i,j}$)
13. $Y_{m,n} \leftarrow Y_{m,n} + S$
14. **end if**
15. Set new promotion event
16. **end for**
17. **if** $B_{i,j}$ is the current active job **then**
18. /* Check if the spare can 'sleep' in the slack of $B_{i,j}$*/
19. $earliest\_promotion \leftarrow$
20. time_to_earliest_promotion_event
21. $\Delta_{ep} \leftarrow earliest\_promotion$
22. **if** $\Delta_{ep} \geq \Delta_{critical}$ **then**
23. Set sleep_exit event at t = time + Δ_{ep}
24. Put spare into sleep mode
25. **end if**
26. **end if**
27. **Event –** Sleep_exit:
28. **if** the upper queue is not empty **then**
29. /* There are backups not yet cancelled */
30. Execute the highest EDF priority job on spare
31. **else**
32. $\Delta_{ep} \leftarrow earliest\_promotion$
33. **if** $\Delta_{er} \geq \Delta_{critical}$ **then**
34. Set wake-up event at t = time + Δ_{er}
35. Put spare to sleep mode
36. **end if**
37. **end if**

if the main job is faulty, we can continue running the spare as scheduled. In lines 26-31 if there is no ready task available for execution, the processor will remain idle. The primary processor will start executing jobs again when the next job arrives. By the use of the task period values, we can compute the earliest release times among all future jobs in linear time. The time to the earliest release time is denoted by the *earliest_release* in the pseudo code. If the idle time exceeds the break to sleep time ($\Delta_{critical}$), the primary processor is put into sleep mode until next arrival (release). In lines 32-34 if there is ready task available for execution, the processor will execute the task.

Algorithm 2 gives the actions taken in response to the events on the spare processor. The spare processor uses adaptive dual-queue scheduling and applies only DPM for energy management. According to lines 1-3 of the algorithm 2, when a job $B_{i,j}$ is promoted to the upper queue it is qualified for execution under EDF scheduling. Lines 4-8 explain that, if a backup job $B_{i,j}$ is completed successfully earlier than the corresponding main task $J_{i,j}$, the execution of remaining part of $J_{i,j}$ will be cancelled on the primary processor. In lines 9-16, if a backup job is entirely cancelled before execution, we compute the slack time that obtains from cancelling $B_{i,j}$. All the tasks that are released until now will be postponed according to the computed slack time. However corresponding deadlines of spare tasks should not be missed. Lines 17-26 express that if a backup job is cancelled but we cannot use slack time to postpone the other spare jobs, therefore, we apply DPM on the processor. The time to earliest promotion event is denoted by variable *earliest_promotion*. If the idle interval is greater than $\Delta_{critical}$, the spare processor is put into the sleep mode and the corresponding sleep-exit event is scheduled. Lines 27-37 explain that at sleep-exit, the spare inspects the upper queue. If the upper queue is empty, by considering the earliest promotion time it switches to the sleep mode. Otherwise, the highest-priority job is executed.

4. Results and Discussion

Our evaluation consists of the comparison between AdDQ and state-of-the-art RAPM. Also we compared AdDQ with the ASSPT, CSSPT and SSFP algorithms (explained in Section 1). To evaluate AdDQ we constructed a discrete-event simulator. In our simulations, for each data point, we generated 1000 task sets and the average results are reported. Each task set consists of 10 tasks. The task periods are generated randomly between 10 and 100 ms. The worst-case utilization of the tasks, are generated randomly using the UUnifast scheme [25]. The worst-case execution time (WC) is computed as the product of the period and worst-case utilization. Like [19] and [20], the actual execution time

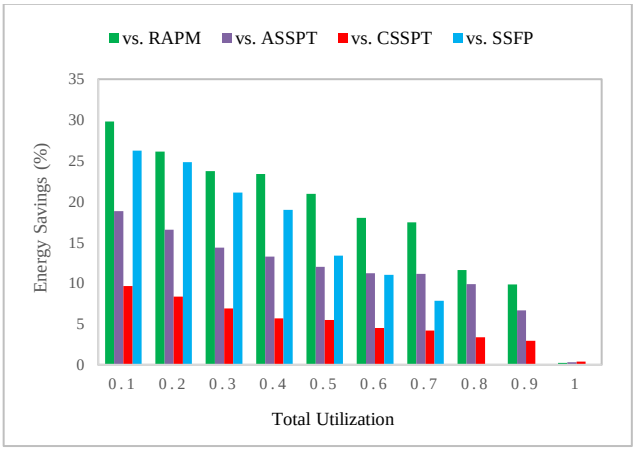

Fig. 4. Energy saving in different system utilizations. Tasks AC are generated based on the uniform distribution and WC/BC=5.

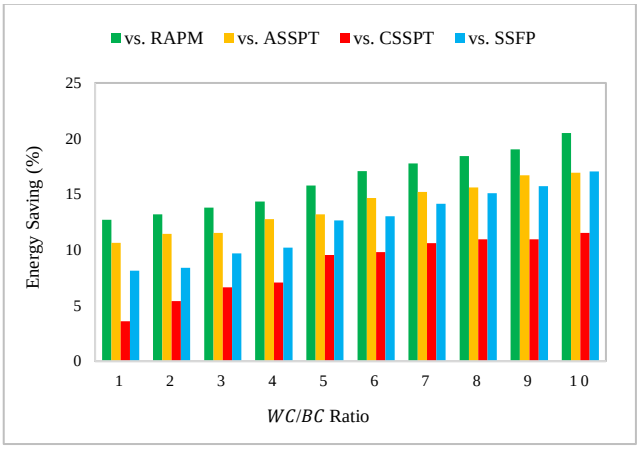

Fig. 6. Energy saving for different workload variability. Tasks AC are generated based on the uniform distribution and U_{tot}=0.5.

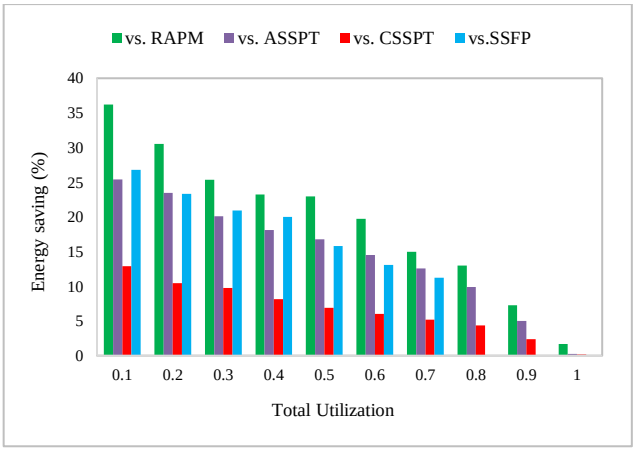

Fig. 5. Energy saving in different system utilizations. Tasks AC are generated based on the normal distribution and WC/BC=5.

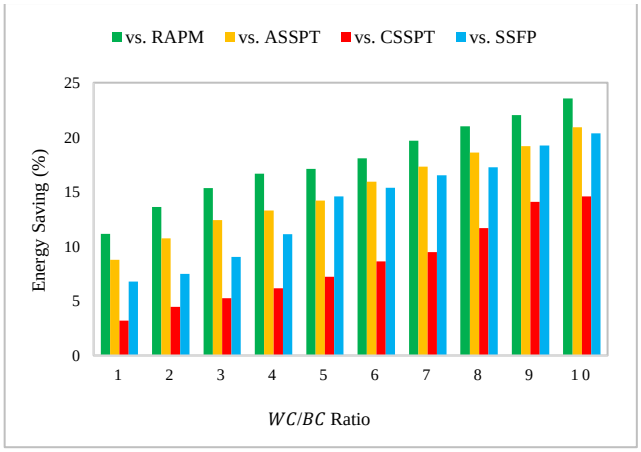

Fig. 7. Energy saving for different workload variability. Tasks AC are generated based on the normal distribution and U_{tot}=0.5.

(AC) of a task instance is obtained randomly according to the uniform distribution or normal distribution with mean (WC+BC)/2 and variance (WC+BC)/6 to ensure that 99.7% of the actual execution times lies within the [BC, WC] range of the task [7] [26]. We evaluated the ratio of energy saving of AdDQ versus RAPM, ASSPT, CSSPT and SSFP across different system parameters including the total utilization (Utot) and ratio between worst-case to best-case execution time (WC/BC). We first evaluated the impact of the system utilization when WC/BC=5 and the total utilization varies from 0.1 to 1. Fig. 4 and Fig. 5 show the results for the cases when tasks actual execution times (AC) are generated based on the uniform and normal distributions. What can be inferred from these figures is that, AdDQ completely outperforms the other four schemes for all utilization values. This is achieved through reducing the overlap of the execution of main tasks on the primary processor and their corresponding backup tasks on the spare processor. Therefore, by further postponing the backup tasks at run time, in many cases we can cancel the backup tasks on the spare processor. Also, for all utilization values AdDQ can save more energy compare to RAPM. The main reason is that RAPM is forced to run at high frequency and consequently consuming too much energy; on the whole, by increasing the utilization, the energy saving decreases (see Fig. 4 and Fig. 5). Since, when utilization is low, more slack time can be achieved, this further slack time helps us to save more energy through DVS and DPM.

In this part we show the impact of workload variability. We set the total utilization to 0.5 and vary the WC/BC ratio from 1 to 10. Fig. 6 and Fig. 7 show the results for the cases when tasks actual execution times (AC) are generated based on the uniform and normal distributions. As the WC/BC ratio increases, there is more dynamic slack for the algorithms to reclaim and the energy consumption decreases correspondingly and the energy saving increases for all schemes. However, AdDQ has better performance against other schemes since it tries to slow down all the jobs in a balanced order.

5. Conclusion

In this paper, we considered two main objectives in designing real-time embedded systems, denoted as reliability and energy consumption. To achieve these objectives we proposed a scheduling scheme for standby-sparing systems that execute periodic real-time tasks. Our scheme uses the EDF scheduling and DPM on the primary processor. On the spare processor, we apply our proposed adaptive dual-queue (AdDQ) scheduling along with DPM. AdDQ postpones the execution of backup tasks on the spare processor as much as possible. Since faults are naturally rare event and also tasks often consume less than their worst-case execution time, tasks commonly complete early or successfully. Another feature of AdDQ is that, it provides a good opportunity to cancel the backup tasks on the spare processor when its main task completes early or successfully on the spare processor; resulting in a reduced energy consumption. We compared our AdDQ with the RAPM, SSFP, ASSPT and CSSPT schemes. Simulation results show that AdDQ provides 14% energy saving compared to the other schemes.

References

[1] S. Aminzadeh, and A. Ejlali, "A Comparative Study of System-Level Energy-Management Methods for Fault-Tolerant Hard Real-Time Systems," *IEEE Trans. on Computers*, vol. 60, no. 9, pp. 1288-1299, 2011.

[2] A. Ejlali, and B.M. Al-Hashimi, "A standby-sparing technique with low energy-overhead for fault-tolerant hard real-time systems," in Proc. of the 7th *IEEE/ACM int'l conf. on Hardware/software codesign and system synthesis* (CODES+ISSS '09), New York, 2009.

[3] T.D. Burd, T.A. Pering, and A.J. Stratakos, "A dynamic voltage scaled microprocessor system," *IEEE J. Solid-State Circuits* (JSSC), vol. 35, no. 11, pp. 1571-1580, 2000.

[4] M. Salehi, and A. Ejlali, "A Hardware Platform for Evaluating Low-Energy Multiprocessor Embedded Systems Based on COTS Devices," *IEEE Trans. on Industrial Electronics*, vol. 62, no. 2, pp. 1262-1269, 2015.

[5] P. Pillai, and K.G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *SOSP ACM Symposium on Operating Systems Principles*, Dec. 2001.

[6] A. Ejlali, B.M. Al-Hashimi, and P. Eles, "Low-Energy Standby-Sparing for Hard Real-Time Systems," *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, vol. 31, no. 3, pp. 329-342, 2012.

[7] H. Aydin, and R. Melhem, "Power-aware scheduling for periodic real-time tasks," *IEEE Trans. on Computers*, vol. 53, no. 5, pp. 584 - 600, May 2004.

[8] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 3, pp. 299,316, 2000.

[9] M. Salehi, A. Ejlali, and B.M. Al-Hashimi, "Two-Phase Low-Energy N-Modular Redundancy for Hard Real-Time Multi-Core Systems," *IEEE Trans. on Parallel and Distributed Systems*, no. 99, 2015.

[10] D. Pradhan, Fault Tolerant Computer System Design, Prentice-Hall, 1996.

[11] D. Zhu, R. Melhem, and D. Mossé, "The effects of energy management on reliability in real-time embedded systems," in *IEEE/ACM int'l Conf. on Computer Aided Design*, Nov. 2004.

[12] I. Koren, and C.M. Krishna, Fault-Tolerant Systems, Morgan Kaufman, 2007.

[13] P. Pop, V. Izosimov, P. Eles, and Z. Peng "Design optimization of time-and cost-constrained fault-tolerant embedded systems with checkpointing and replication," IEEE Trans. on Very Large Scale Integration (VLSI) Systems, vol. 17, no. 3, pp. 389-402, 2009.

[14] R. Melhem, D. Mosse, and E. Elnozahy, "The Interplay of Power Management and Fault Recovery in Real-Time Systems," *IEEE Trans. on computers*, vol. 53, pp. 217-231, 2004.

[15] B. Zhao, H. Aydin, and D. Zhu, "Enhanced reliability-aware power management through shared recovery technique," in Proc. int'l conf. on Computer Aided Design (ICCAD), 2009.

[16] D. Zhu, and H. Aydin, "Reliability-Aware Energy Management for Periodic Real-Time Tasks," IEEE Trans. on Computers, vol. 58, no. 10, pp. 1382-1397, April 2009.

[17] R. Sridharan, and R. Mahapatra, "Reliability aware power management for dual-processor real-time embedded systems," in Proc. of the 47th Design Automation Conference (DAC), New York, 2010.

[18] M. Khavari Tavana, M. Salehi, and A. Ejlali, "Feedback-Based Energy Management in a Standby-Sparing Scheme for Hard Real-Time Systems," in Proc. of the 32nd IEEE Real-Time Systems Symposium, RTSS, Vienna, 2011.

[19] M.A. Haque, H. Aydin, and D. Zhu, "Energy-Aware Standby-Sparing Technique for Periodic Real-Time Applications," *Proc. IEEE 29th Int'l Conf. Comput. Design (ICCD'11),* pp. 190-197, Oct. 2011.

[20] M.A. Haque, H. Aydin, and D. Zhu, "Energy Management of Standby-Sparing Systems for Fixed-Priority Real-Time Workloads," *Green Computing Conf. (IGCC)*, Arlington, June 2013.

[21] C.L. Liu, and W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," Journal of the ACM (JACM), vol. 20, no. 1, pp. 46-61, Jan. 1973.

[22] R. Davis, and A. Wellings, "Dual Priority Scheduling," *in the 16th IEEE Real-Time Systems Symposium*, Pisa, 1995.

[23] J. Lehoczky, L. Sha, and Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," in *Proc. of IEEE Real Time Systems Symposium*, 1989.

[24] R. Gupta, "Dynamic Voltage Scaling for Systemwide Energy Minimization in Real-Time Embedded Systems," in *Proc. of the int'l Symposium on Low Power Electronics and Design*, Newport Beach, Aug. 2004.

[25] E. Bini, and G.C. Buttazzo, "Measuring the Performance of Schedulability Tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129-154, 2005.

[26] F. Gruian, "Hard real-time scheduling for low-energy using stochastic data and DVS processors," in *Proc. of IEEE int'l Symposium on Low power electronics and design (ISLPED)*, 2001.

[27] F. R. Poursafaei, S. Safari, M. Ansari, M. Salehi ,and A. Ejlali "Offline replication and online energy management for hard real-time multicore systems." in *Proc. of IEEE int'l Symposium on Real-Time and Embedded Systems and Technologies (RTEST)*, pp. 1-7, 2015.

Mohsen Ansari received the M.Sc. degree in computer engineering from Sharif University of Technology, Tehran, Iran, in 2016. He is currently working toward the PhD degree in computer engineering form Sharif University, Tehran, Iran. He is now the member of Embedded Systems Research Laboratory (ESR-LAB) at the department of computer engineering, Sharif University of Technology. His research interests include low-power design of embedded systems, peak power management in fault-tolerant embedded systems, and multi-/many-core systems with a focus on dependability/reliability.
Email: mansari@ce.sharif.edu

Sepideh Safari received the M.Sc. degree in computer engineering from Sharif University of Technology, Tehran, Iran, in 2016. She is currently pursuing her PhD degree with Sharif University of Technology, Tehran, Iran. She is now the member of VLSI Laboratory at the department of computer engineering, Sharif University of Technology. Her research interests include

low-power design, multi-/many-core systems, and energy management in fault-tolerant real-time systems.
Email: ssafari@ce.sharif.edu

Farimah R. Poursafaei received her B.Sc. degree in computer engineering from Amirkabir University of Technology, and her M.Sc. degree from Sharif University of Technology, in 2016, Tehran, Iran. Her research interests include multi-/many-core systems, low-power design, real-time embedded systems, and optimization and management of non-volatile memories.
Email: poursafaei@ce.sharif.edu

Mohammad Salehi received the PhD degree in computer engineering from Sharif University of Technology, Tehran, Iran, in 2016. He is currently an assistant professor of computer engineering at University of Guilan, Rasht, Iran. From 2014 to 2015, he was a visiting researcher in the Chair for Embedded Systems (CES), Karlsruhe Institute of Technology (KIT), Germany. His research interests include design of low-power, reliable and real-time embedded systems with a focus on dependability and energy efficiency in cyber-physical systems and Internet of Things (IoT).
Email: mohammad.salehi@guilan.ac.ir

Alireza Ejlali received the PhD degree in computer engineering from Sharif University of Technology in, Tehran, Iran, in 2006. He is currently an associate professor of computer engineering at Sharif University of Technology. From 2005 to 2006, he was a visiting researcher in the Electronic Systems Design Group, University of Southampton, Southampton, United Kingdom. In 2006, he joined Sharif University of Technology as a faculty member in the department of computer engineering and from 2011 to 2015 he was the director of Computer Architecture Group in this department. His research interests include low power design, real-time embedded systems, and fault-tolerant embedded systems.
Email: ejlali@sharif.edu

Paper Handling Data:

Submitted: 15.04.2017
Received in revised form: 10.04.2018
Accepted: 27.04.2018
Corresponding author: Dr. Alireza Ejlali,
Department of Computer Engineering, Sharif University of Technology, Tehran, Iran.