Reliability-Aware Scheduling for Periodic Tasks Requiring (m,k)-firm Real-Time Data Processing

Linwei Niu
Department of Electrical Engineering and Computer Science
Howard University
Washington, DC, U.S.A.
linwei.niu@howard.edu

***Abstract*—For real-time systems, reliability and QoS (Quality of Service) are among the primary design concerns. In this research, we proposed a reliability-aware scheduling scheme for periodic tasks requiring (m,k)-firm real-time data processing. The (m,k)-firm real-time constraint requires that at least m out of any k consecutive jobs of a periodic task meet their deadlines. To achieve the dual goals of maximizing the system reliability and QoS while satisfying the feasibility requirement, we propose to reserve recovery space for real-time jobs in an adaptive way based on the mandatory/optional job partitioning strategy. Moreover, advanced techniques are studied to implement task assignment without affecting the system feasibility. The evaluation results demonstrate that the proposed techniques significantly outperformed the previous research in maximizing the reliability and QoS for (m,k)-firm real-time systems while preserving the system feasibility.**

***Index Terms*—Reliability, QoS, feasibility**

I. Introduction

In traditional hard real-time systems, all task instances are required to meet their deadlines and any deadline miss will crash the entire application or system. However, in many practical real-time applications such as multimedia data processing and real-time communication systems, occasional deadline misses can often be tolerated. Some other applications may have soft deadlines where tasks that do not finish by their deadlines can still be completed with a reduced value [7] or they can simply be dropped provided that the user's perceived quality of service (QoS) is satisfied.

QoS requirements dictate under what conditions the system will provide its service to real-time tasks executed in the system. To quantify the QoS requirements, some statistic information such as the average deadline miss rate can be used. However, even a low overall miss rate (e.g., 2%) cannot prevent such kind of situation that a very large number of deadline misses (e.g., 20 deadline misses out of 1000 jobs) occur consecutively in a short period of time, which could generate undesirable results.

The deterministic QoS model is more appropriate for such kind of systems. Under the deterministic QoS model, tasks have both firm deadlines (i.e., task(s) with deadline(s) missed generate(s) no useful values) and a throughput requirement (i.e., *sufficient* task instances must finish before their deadlines to provide acceptable QoS levels) [8]. Two well known deterministic QoS models are the (m,k)-model [2] and the *window-constrained* model [13]. The (m,k)-model requires that m jobs out of any *sliding* window of k consecutive jobs of the task meet their deadlines, whereas the *window-constrained* model requires that m jobs out of each *fixed* and *nonoverlapped* window of k consecutive jobs meet their deadlines. It is not hard to see that the *window-constrained* model is weaker than the (m,k)-model as the latter one is more restrictive.

To ensure the (m,k)-constraints, Ramanathan *et al.* [12] proposed a partitioning strategy which divides the the jobs into *mandatory* ones and *optional* ones. The mandatory ones are jobs that must meet their deadlines in order to satisfy the (m,k)-constraints. In other words, so long as all the mandatory jobs can meet their deadlines, the (m,k)-constraints can be ensured. In [13], West *et al.* tried to set up a corresponding relationship from the *window-constrained* model to the (m,k) model. They also found a method to map the *window-constraints* to the (m,k)-constraints.

In recent years, reliability has become increasingly important in the design of fault tolerant computer systems as system fault could occur anytime during the execution cycle of real-time jobs [19]. In order to satisfy the reliability requirement of a given job, a widely adopted strategy is to reserve a recovery job for it. If the job has failed due to transient faults [4], the recovery job should be invoked for execution to compensate the failed job.

In the context of reliability assurance, the issue of system feasibility is especially critical as the recovery job(s) (for preserving the system reliability) will also occupy part of the system utilization, which could affect the feasibility of the system adversely. When the QoS is taken into consideration, the problem becomes even more challenging as we need to ensure that the baseline (m,k)-constraints be satisfied all the time without exception.

In this paper, we study the problem of maximizing the reliability and QoS for periodic real-time tasks with (m,k)-constraint in data processing while preserving the system feasibility.

The rest of the paper is organized as follows. Section II presents the preliminaries. Section III presents the motivations. Section IV presents our general scheduling algorithm. In section V, we present our evaluation results. In section VI

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

we offer the conclusions.

II. Preliminaries

A. System models

The real-time task set considered in this paper contains n independent periodic tasks, $\mathcal{T} = \{\tau_1, \tau_1, \cdots, \tau_n\}$, scheduled according to the earliest deadline first (EDF) policy [6]. Each task contains an infinite sequence of periodically arriving instances called *jobs*. Task τ_i is characterized using five parameters, *i.e.*, (P_i, D_i, C_i, m_i, k_i). P_i, D_i ($D_i \leq P_i$), and C_i denote the period, the deadline and the worst case execution time for τ_i, respectively. A pair of integers, *i.e.*, (m_i,k_i) $(0 < m_i \leq k_i)$, are used to represent the (m,k)-constraint for task τ_i which requires that, among any k_i consecutive jobs, at least m_i jobs are executed successfully. The j^{th} job of task τ_i is denoted as J_{ij} and its arrival time and absolute deadline are denoted as r_{ij} and d_{ij}, respectively. In addition, if job J_{ij} is used as a recovery job, it is also represented as $J_{ij}(R)$. The hyper-period of the task set, represented by H, is defined as $H = LCM(k_iP_i)$.

We assume the task set is to be executed in a uni-processor system.

B. Fault and Recovery Models

Similar to [3], [9], we focus on transient faults in this paper since transient faults occur much more frequently than permanent faults in modern semi-conductor devices [1]. We assume that faults can be detected using sanity (or consistency) checks [10] when a job finishes its execution and the overhead for detection can be integrated into the job's worst case execution time. Moreover, when transient fault occurs and is detected at the end of a job's execution, the affected job can be addressed by re-executing a recovery job with the same worst-case execution time as the original one [10]. Once released, the recovery job will be executed like a normal job.

C. Performance Metrics

1) Reliability: Following the fault model in [19], we assume that the transient faults will present Poisson distribution [15] and the average transient fault rate for systems running at full speed is σ. Based on it, for any job J_{ij} of task τ_i, the reliability of it, represented by $\gamma(J_{ij})$, is defined as the probability that J_{ij} could be completed successfully at full speed. According to [19], $\gamma(J_{ij})$ is given as:

$$\gamma(J_{ij}) = e^{-\sigma C_i} \tag{1}$$

Under the QoS-constraint (either in the (m,k)-constraint of (m_i,k_i), or the window-constraint of m_i/k_i), the reliability of the l^{th} window W_{il} of task τ_i, represented by $\gamma(W_{il})$, is defined as the probability that at least m_i jobs in W_{il} could be completed successfully.

Based on it, the reliability of task τ_i is defined as,

$$\gamma(\tau_i) = \prod_{l=1}^{z_i} \gamma(W_{il}) \tag{2}$$

where $z_i = \frac{H}{k_iP_i}$.

And the reliability of the whole system $\mathcal{T}$ is defined as,

$$\gamma(\mathcal{T}) = \prod_{i=1}^{n} \gamma(\tau_i) \tag{3}$$

2) Quality of Service: Based on the above system models, we define the metrics to measure the quality of service of a task τ_i, represented by $QoS(\tau_i)$ to be the ratio of the number of jobs completed successfully and the total number of jobs within the hyperperiod H. Specifically, $QoS(\tau_i)$ can be calculated as followed:

$$QoS(\tau_i) = \frac{\Sigma_{l=1}^{z_i}(m_i \times \gamma(W_{il}))}{z_i \times k_i} = \frac{m_i}{k_i} \times \frac{\Sigma_{l=1}^{z_i}\gamma(W_{il})}{z_i} \tag{4}$$

And the system quality of service is defined as

$$QoS_{sys} = \sum_{i=1}^{n} QoS(\tau_i)\omega_i \tag{5}$$

where ω_i is the user-defined weight for task τ_i.

III. Motivations

To satisfy the reliability requirement, one essential part is to reserve recovery jobs for the tasks when applying DVFS to reduce energy. Prior to that, a key problem for ensuring the (m,k)-constraints is to judiciously partition the jobs into *mandatory* jobs and *optional* jobs [11]. A well-known partitioning method is called the *the evenly distributed pattern* (or E-pattern) [12]. According to E-pattern, the pattern π_{ij} for job J_{ij}, *i.e.*, the j^{th} job of a task τ_i, is defined by:

$$\pi_{ij} = \begin{cases} \text{“1”} & \text{if } j = \lfloor \lceil \frac{(j-1)\times m_i}{k_i} \rceil \times \frac{k_i}{m_i} \rfloor + 1 \\ \text{“0”} & \text{otherwise} \end{cases} \qquad j = 1, 2, \cdots \tag{6}$$

where “1” represents the mandatory job and “0” represents the optional job. In [5], a variation of E-pattern called E^R-pattern was achieved by reversing the pattern horizontally to let the optional jobs happen first, which can preserve the schedulability of E-pattern [5].

Note that based on the E-pattern given in Equation (6), without energy management, *i.e.*, all mandatory jobs to be executed at full speed without recovery, the reliability of any window W_{il} of k_i jobs could be calculated as:

$$\gamma(W_{il}) = \prod_{p=(l-1)\times k_i+1}^{(l-1)\times k_i+k_i} \{\gamma(J_{ip}) \mid \pi_{ip} = 1\} \tag{7}$$

It is also noted that the job patterns defined with E-pattern have the property that they define a *minimal set* of mandatory jobs that “just” satisfies the given (m,k)-constraint in each sliding window. Due to this property, before the speed for any task is scaled to save energy, a popular approach is to reserve a recovery job for each mandatory job of the task to satisfy the reliability of the same task. Conversely, under window-constraint, this requirement could be greatly relaxed as we only need to reserve *one* recovery job within each separate window so long as the reliability of the window under consideration is not lower than that calculated with Equation (7). However,

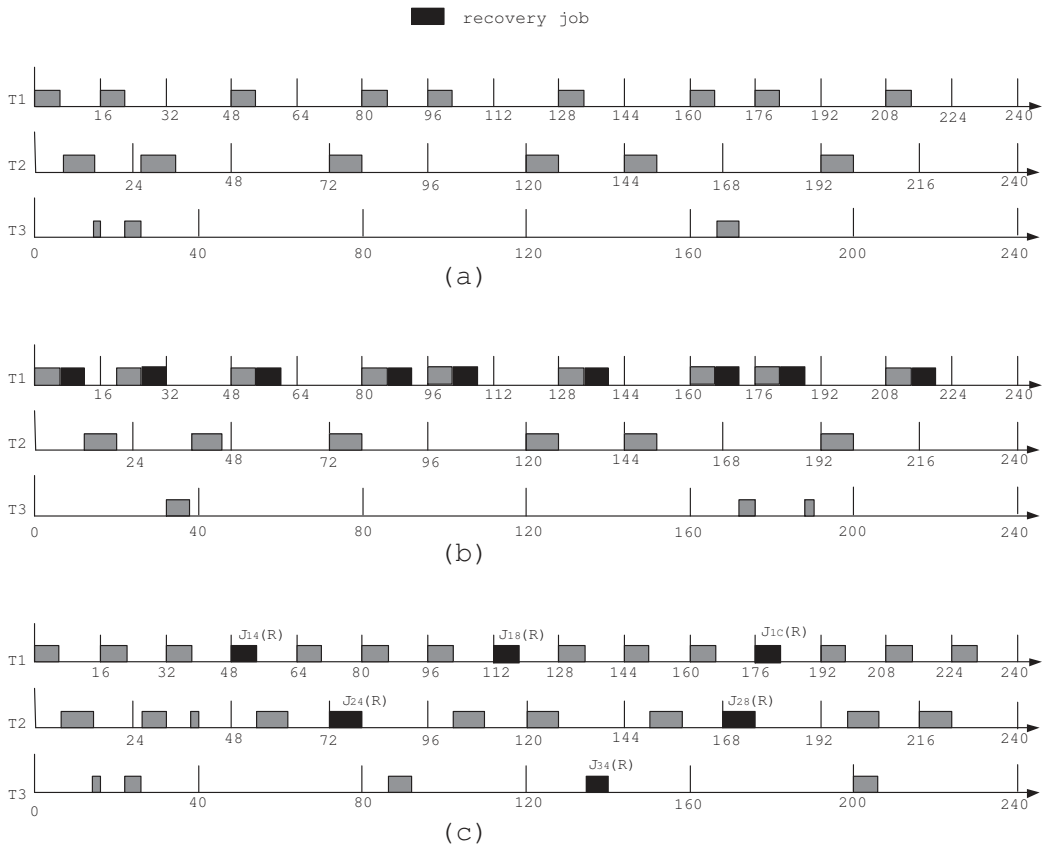

Fig. 1. (a) The schedule for task set $\{\tau_1 = (16,16,6,3,5);\ \tau_2 = (24,24,8,3,5);\ \tau_3 = (40,40,6,2,8)\}$ with no recovery jobs reserved; (b) The schedule for the same task set with the state-of-the-art recovery job reservation strategy based on E-pattern (can only accommodate recovery jobs for task τ_1); (c) The schedule with recovery jobs reserved based on *window-constraints* that could be transferred to their original (m,k)-constraints.

the problem is, since window-constraint is weaker than (m,k)-constraint, a task τ_i satisfying the window-constraint of m_i/k_i does not necessarily satisfy the (m,k)-constraint of (m_i,k_i). Fortunately, in [13] it is shown that the *window-constraint* of m_i/k_i can be transferred to the (m,k)-constraint of $(m_i, 2k_i - m_i)$. That means if we can reserve recovery jobs for the tasks such that it could satisfy the window constraint of m_i/k_i, it will satisfy the (m,k)-constraint of $(m_i, 2k_i - m_i)$ automatically. Since in this case we only need to reserve one recovery job for each separate window of k_i jobs, it could leave us more space to keep the energy under control than reserving recovery jobs for all mandatory jobs under E-pattern for tasks with (m,k)-constraints of $(m_i, 2k_i - m_i)$, which could be illustrated using the following example.

Consider a task set of three tasks, *i.e.*, $\tau_1 = (16,16,6,3,5)$, $\tau_2 = (24,24,8,3,5)$, and $\tau_3 = (40,40,6,2,8)$. Figure 1(a) shows the schedule for the mandatory jobs based on E-pattern. If we assume the probability of transient fault at full speed to be 10^{-6}, without reliability management, based on Equation (7), the reliability of each window in τ_1, τ_2, and τ_3 will be 0.9999820002, 0.9999760003, and 0.9999880001, respectively. Based on Equation (3), the corresponding system reliability will be 0.9999460015. Meanwhile, if we assume all tasks in Equation (5) have equal weights, *i.e.*, $\omega_i = \frac{1}{n}$ for all tasks in $\mathcal{T}$, the QoS of the whole system based on Equation (5) will be 0.48332393344.

If we apply the state-of-the-art reliability management strategy, *i.e.*, reserve a recovery job for each mandatory job whenever possible to do so, Figure 1(b) shows the schedule for the mandatory jobs based on E-pattern for the original given (m,k)-constraints, with a recovery job reserved for each mandatory job of task τ_1. Note that in this case after reserving recovery jobs for task τ_1, there is not enough room for reserving recovery jobs for task τ_2 or task τ_3 any more. Therefore, only the reliability of each window in τ_1 could be improved to 0.99999999996 while the reliability of each window in task τ_2 and τ_3 did not change. As a result, the reliability of the whole system based on Equation (3) is now 0.99996400056. And the QoS of the whole system in this case based on Equation (5) is 0.48332753339. Compared with the above schedule in Figure 1(a) without reliability management, there is some improvement in the system reliability and very slight improvement in system QoS.

However, if we follow a different way of managing the reliability, the system performance could be improved substantially.

As shown in Figure 1(c), different from the above schedule, if we reserve the recovery jobs to satisfy the window-constraints first, we can still meet the original (m,k)-constraints with opportunities to have more tasks managed with recovery jobs, thereby improving the system reliability further. As illustrated in Figure 1(c), in this case the mandatory jobs of τ_1, τ_1, and τ_3 are determined to satisfy the window constraints of $3/4$, $3/4$, and $2/5$ first. Then according to the aforementioned mapping relationship they can meet their original (m,k)-constraints of $(3,5)$, $(3,5)$, and $(2,8)$, respectively. Based on the above configuration, with the aforementioned flexibility of window-constraints in preserving reliability in each separate window, we only need to reserve one recovery job for each window under consideration. For ease of presentation here we assume the next optional job following the last mandatory job in the same window is reserved as recovery job for each separate window, as shown in Figure 1(c). Under the new schedule, the reliability of each window W_{il} of task τ_i should be calculated as:

$$\begin{aligned}\gamma(W_{il}) &= \prod_{p=(l-1)k_i+1}^{(l-1)k_i+k_i} \{\gamma(J_{ip}) \mid \pi_{ip}=1\} \\ &+ \sum_{p=(l-1)k_i+1}^{(l-1)k_i+k_i} \{ \prod_{q=(l-1)k_i+1}^{(l-1)k_i+k_i} \{\gamma(J_{iq}) \mid \pi_{iq}=1,\ q\neq p\} \\ &\times (1-\gamma(J_{ip}))\times\gamma(J_{ip})\} \end{aligned} \quad (8)$$

Based on Equation (8), the reliability of each window in task τ_1, τ_2, and τ_3 will be calculated as 0.99999999978, 0.9999999996, and 0.99999999989, respectively. Based on Equation (3), the corresponding system reliability will be improved to 0.99999999927, which is much higher than that under the state-of-the-art reliability management strategy. In addition, the system QoS based on Equation (5) in this case is improved to 0.63333333316, which is 20.7% higher than that in Figure 1(b) (and that in Figure 1(a) without reliability management as well).

From the above example, it is not hard to see that it is very promising to determine the job/recovery patterns in such a way that the task set satisfies the window-constraint first which could be transferred to the corresponding original (m,k)-constraint automatically. However, since the system might not be able to accommodate recovery jobs for all tasks, how to select the subset of tasks that can be managed with recovery jobs will be an important issue. Regarding that, in this paper a method based on "branch-and-bound" is proposed to solve this issue systematically.

IV. The General Algorithm

In this section, we will introduce our general scheduling algorithm. Our algorithm consists of two phases: an off-line phase and an on-line phase.

A. The Off-Line Phase

The goal for the off-line phase is to determine the (shared) recovery jobs for the tasks such that the expected reliability and QoS of the system are maximized while the feasibility of the task set is preserved. One essential part of it is to determine the mandatory jobs for the tasks and reserve recovery jobs for them based on the window-constraints (that can be transferred to the corresponding original (m,k)-constraints) if possible. However, since the system might not be able to accommodate recovery jobs for all tasks, how to select the subset of tasks that can be managed with recovery jobs is not a trivial problem. Then the problem becomes how to select the subset of tasks to be managed with recovery jobs to achieve the best system reliability and QoS. In [18], it is shown that even without consideration of reliability and QoS this problem is NP-hard. Although some heuristics [18] are proposed for hard real-time systems regarding the selection of tasks, they are not applicable any more for soft real-time systems with (m,k)-constraints. In order to solve the problem, in this section, we propose a "branch-and-bound" method to divide the task set $\mathcal{T}$ into two parts: the subset Ω in which the tasks will be

Algorithm 1 Reserving recovery jobs for the task set

Input: task set $\mathcal{T}$ with mandatory jobs determined by E-pattern (with no recovery job(s) reserved for any task yet);
Output: task set Γ=$\Omega\cup\Theta$, where Ω is the subset of tasks in $\mathcal{T}$ with recovery jobs reserved for each task and Θ is the subset of tasks in $\mathcal{T}$ with no recovery jobs;
$\Omega = \emptyset$;
$\Theta =$ all tasks in $\mathcal{T}$;
// no recovery jobs is reserved yet;
Γ=$\Omega\cup\Theta$;
Compute the system reliability and QoS, *i.e.*, $\gamma(\mathcal{T})$ and QoS_{sys} of the task set $\mathcal{T}$ under original (m,k)-constraints based on Equations (3) and (5);
$Bound = \gamma(\mathcal{T}) \times QoS_{sys}$;
Recovery-Reservation (Ω, Θ, *Bound*, Γ);
output (Γ);

FUNCTION Recovery-Reservation(Ω, Θ, *Bound*, Γ)
for each task $\tau_i \in \Theta$ **do**
  Re-determine the mandatory jobs of τ_i using E-pattern based on the window-constraint that can be transferred to its original (m,k)-constraint;
  Remove τ_i from Θ;
  Add τ_i to Ω;
  In each separate window of task τ_i, reserve the next optional job following the last mandatory job in the same window as recovery job;
  Apply the *LPEDF* algorithm from [14] to the task set $\Omega\cup\Theta$;
  if $\Omega\cup\Theta$ is feasible with *LPEDF* **then**
    Compute the system reliability and QoS, *i.e.*, $\gamma(\mathcal{T})$ and QoS_{sys} of the updated task set $\Omega\cup\Theta$ based on Equations (3) and (5);
    $newBound = \gamma(\mathcal{T}) \times QoS_{sys}$;
    if $newBound > Bound$ **then**
      $Bound = newBound$;
      Γ=$\Omega\cup\Theta$;
    end if
    Recovery-Reservation (Ω, Θ, *Bound*, Γ);
  else
    Restore the mandatory jobs of τ_i to the original ones based on E-pattern (with no recovery job(s)) and put it back to Θ;
  end if
end for

managed with recovery jobs and the subset Θ in which the tasks will not be managed with recovery jobs. The details are presented in Algorithm 1.

As can be seen in Algorithm 1, by applying the branch-and-bound strategy, our approach determines task by task if the mandatory jobs of each task should be based on the original (m,k)-constraint (with no recovery jobs reserved) or based on the window-constraint (with recovery jobs reserved according to Line 17). When Algorithm 1 is finished, it is possible to reach certain hybrid configuration in which the tasks in Ω are partitioned based on window-constraints with recovery jobs reserved in each separate window, while the tasks in Θ are still partitioned based on the original (m,k)-constraints with no recovery jobs reserved for them. And the resulting configuration should be the one that can maximize the system reliability-QoS product while preserving the overall task set feasibility.

Note that in lines 18-19 of Algorithm 1, Yao's LPEDF Algorithm [14] is applied to check the feasibility of the mandatory jobs in each iteration.

Algorithm 2 Online algorithm

```
1:  Upon job arrival:
2:  Run jobs in the ready queue according to EDF scheme with their
    predetermined speed;
3:
4:  Upon job completion:
5:  if the execution of J_i is found to have failed then
6:      if J_i ∈ Θ and J_i has a recovery job reserved for it then
7:          Invoke J_i's recovery job immediately;
8:      else
9:          if J_i ∈ Ω and J_i is not a recovery job then
10:             Invoke J_i's recovery job in the same window at its arrival
                time;
11:         end if
12:     end if
13: else
14:     if J_i ∈ Ω and the window which J_i belongs to has got enough
        number of jobs completed successfully then
15:         Drop the recovery job of the window which J_i belongs to;
16:     end if
17: end if
```

Based on the task assignment output by Algorithm 1, the mandatory jobs of each task can be scheduled according to the EDF scheme during the online phase.

B. The Online Phase

During the online phase, as shown in Algorithm 2, a job ready queue will be maintained. Upon arrival, a mandatory job determined during the off-line phase is inserted into the ready queue. Note that a recovery job needs to be executed only if some mandatory job within the same window has failed. Otherwise the recovery job will simply be dropped. All jobs in the ready queue will be executed following the EDF scheme. If the current job J_i is found to have failed at its completion time, its recovery job in the same window needs to be invoked and inserted into the ready queue at its arrival time (line 10).

If the execution of J_i is successful, Algorithm 2 determines whether its shared recovery job in the window it belongs to, if any, should be dropped depending on the number of successful jobs so far in that window (lines 13-15).

The complexity of Algorithm 2 mainly come from scheduling the mandatory/recovery jobs in the ready queue, which is $O(N)$.

V. Evaluation

In this section, we evaluate the performance of our approach by comparing with the existing approaches in literature. Specifically, the performance of three different approaches were studied:

- *MKNR* The task sets are partitioned with E-pattern, and all mandatory jobs are executed without recovery jobs reserved for them.
- *MKR* The task sets are partitioned with E-pattern to satisfy the given (m,k)-constraints. Then the mandatory jobs are scheduled with recovery job reserved for them with the approach similar to that in [18], *i.e.*, reserving recovery jobs (for all mandatory jobs of the same task) for as many tasks as possible.
- *WCMKR* This is our newly proposed approach in Section IV based on window transferring.

The periods of the real-time task sets were randomly chosen in the range of $[30ms, 100ms]$. The worst case execution time (WCET) was set to be uniformly distributed and the m_i and k_i for the user defined (m,k)-constraints were also randomly generated such that k_i is uniformly distributed between 2 to 10, and $m_i < k_i$. To investigate the performance of the different approaches under different workload, we divided the total (m,k)-utilization, *i.e.*, $\sum_i \frac{m_i C_i}{k_i P_i}$, into intervals of length 0.1. To reduce the statistical errors, we required that each interval contain at least 20 task sets schedulable with E-pattern, or until at least 5000 task sets within each interval had been generated.

For the fault and recovery model we adopted the same model as used in [19], *i.e.*, the transient faults are assumed to follow the Poisson distribution with an average fault rate of $\sigma = 10^{-6}$ at full speed.

Firstly, we inspected the system reliability by the different approaches. Here in order to reflect the system reliability in a more straightforward way, we checked the *probability of dynamic failure* (denoted as PoDF) of the different approaches. The PoDF is defined as the probability for any window of k_i consecutive jobs to have less than m_i jobs out of it completed successfully. The result is shown in Figure 2(a).

As seen, in most utilization intervals, the PoDF of *MKR* is lower than that by *MKNR*. The effect is especially obvious when the utilization is relatively low. That conforms to the conclusion in [17] that the reservation of recovery jobs could generally help reduce the probability of job failure. The average PoDFs of *WCMKR* is still much lower than that by *MKR* when the utilization is not very low or not very high. This is mainly because, by reserving shared recovery jobs based on window-constraints (that can be transferred to the corresponding original (m,k)-constraints) first, the mandatory

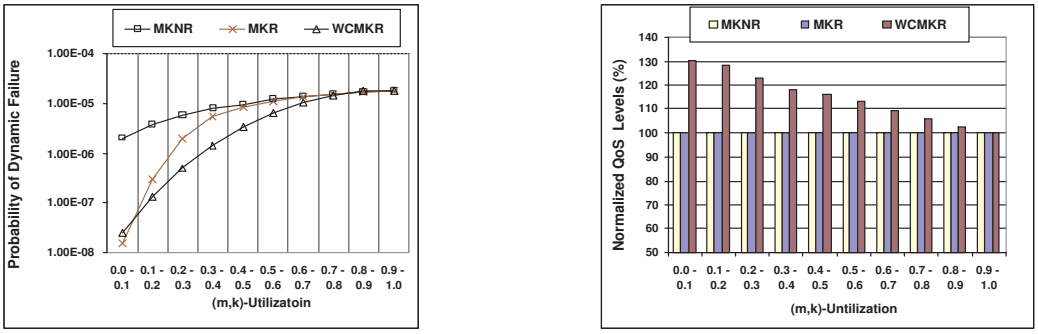

Fig. 2. (a) The comparison on the probability of dynamic failure (PoDF) for the different approaches; (b) The comparison on the QoS for the different approaches.

jobs of more tasks could have recovery jobs reserved for them when compared with *MKR*.

Next, we inspect the system QoS levels each approach can provide. For simplicity, we assumed all tasks in Equation (5) were assigned the equal weights, *i.e.*, $\omega_i = \frac{1}{n}$ for any task $\tau_i \in \mathcal{T}$. The results (normalized to that by *MKNR*) are shown in Figure 2(b). From Figure 2(b), we can see that the newly proposed approach, *i.e.*, *WCMKR* can achieve much better QoS levels than the previous approaches. Compared with *MKR* and *MKNR*, the maximal QoS improvement could be nearly 30%. This is because, different from *MKR* and *MKNR* which could only provide a minimum set of jobs that "just" satisfied the (m,k)-constraints, *WCMKR*, by adopting more adaptive recovery job reservation techniques, could utilize the time space more efficiently. Therefore it could generally accommodate more valid jobs in its schedule, generating better QoS levels.

VI. Conclusions

In this paper, we explored maximizing the reliability and QoS for periodic real-time tasks with (m,k)-constraint in data processing, which requires that at least m out of any k consecutive jobs of a task meet their deadlines. Regarding that, we proposed a reliability-aware scheduling scheme which reserved recovery space for periodic real-time tasks in an adaptive way based on the mandatory/optional job partitioning strategy. Moreover, advanced techniques are studied to implement task assignment without affecting the system feasibility. Through extensive simulations, our evaluation results demonstrate that the proposed techniques significantly outperformed the previous research in reliability and QoS performance while preserving the system feasibility.

References

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

[2] M. Hamdaoui and P. Ramanathan. A dynamic priority assignment technique for streams with (m,k)-firm deadlines. *IEEE Transactions on Computes*, 44:1443–1451, Dec 1995.

[3] Q. Han, L. Niu, G. Quan, S. Ren, and S. Ren. Energy efficient fault-tolerant earliest deadline first scheduling for hard real-time systems. *Real-Time Syst.*, 50(5-6):592–619, Nov. 2014.

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

[5] Linwei Niu and Gang Quan. Energy minimization for real-time systems with (m,k)-guarantee. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 14(7):717–729, July 2006.

[6] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 17(2):46–61, 1973.

[7] J. Liu. *Real-Time Systems*. Prentice Hall, NJ, 2000.

[8] L. Niu and G. Quan. Peripheral-conscious energy-efficient scheduling for weakly hard realctime systems. *International Journal of Embedded Systems*, 7(1):11–25, 2015.

[9] L. Niu and J. Xu. Improving schedulability and energy efficiency for window-constrained real-time systems with reliability requirement. *Journal of Systems Architecture*, 61(5):210–226, May 2015.

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

[11] G. Quan and X. Hu. Enhanced fixed-priority scheduling with (m,k)-firm guarantee. In *RTSS*, pages 79–88, 2000.

[12] P. Ramanathan. Overload management in real-time control applications using (m,k)-firm guarantee. *IEEE Trans. on Paral. and Dist. Sys.*, 10(6):549–559, Jun 1999.

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

[14] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *AFCS*, pages 374–382, 1995.

[15] Y. Zhang, K. Chakrabarty, and V. Swaminathan. Energy-aware fault tolerance in fixed-priority real-time embedded systems. In *ICCAD*, 2003.

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

[17] D. Zhu and H. Aydin. Energy management for real-time embedded systems with reliability requirements. In *ICCAD*, 2006.

[18] D. Zhu and H. Aydin. Reliability-aware energy management for periodic real-time tasks. *Computers, IEEE Transactions on*, 58(10):1382–1397, 2009.

[19] D. Zhu, R. Melhem, and D. Mosse. The effects of energy management on reliability in real-time embedded systems. In *ICCAD*, 2004.