Low Power Round-Robin Scheduling

Raul Brito - Nicolas Navet
Laboratoire LORIA - équipe TRIO
Campus Scientifique - BP 236
54506 Vandoeuvre-lès-Nancy - France
Raul.Brito@loria.fr - Nicolas.Navet@loria.fr

Abstract

Energy consumption is becoming a crucial issue in the design of digital systems especially when considering portable and embedded systems due to their operational dependency on batteries. Since the processor is a major source of energy consumption, energy-aware scheduling strategies that decrease the CPU speed when possible enable to achieve significant energy savings.

In this paper, we study the low-power scheduling under the Round-Robin policy which is widely available since it is part of the Posix1003.1b standard. An algorithm that computes the minimum processor speed for scheduling a job set under Round-Robin is provided. It relies on an efficient feasibility test that is also a contribution of this paper. Finally, we present mechanisms that are necessary for ensuring schedulability at run-time and that reduce consumption when jobs do not require their worst-case execution time.

A counter-intuitive result shown in this study is that a job set might not be feasible at maximum frequency while being feasible with a lower frequency. This implies that even without interests in energy saving, lower frequencies have to be considered for feasibility.

Contents

1. Introduction
2. Model of the system
3. Feasibility under Round-Robin
4. Energy reduction
5. Conclusion

keywords

Real-Time Systems, Scheduling, Round-Robin, Dynamic Voltage Scaling, Schedulability Analysis.

1 Introduction

Context of the paper. With the advent of embedded and portable systems, energy consumption has become a crucial issue in systems design. All battery operated systems, such as PDAs, laptops and mobile phones or even implantable biomedical systems would benefit from a better energy efficiency.

Focusing on the processor, new low level techniques explore the fact that a linear voltage and frequency decrease results in, at least, a quadratic energy consumption reduction [5]. Without any timing constraints, the best solution with regard to (w.r.t.) consumption is to execute the application at the lowest available processor speed which, in general, will not enable the system to meet the required performance level. Setting deadlines is a way to ensure this minimum performance level even on non-real time systems and studying the scheduling with feasibility concern is useful even outside the traditional context of real-time computing.

Until recently the Round-Robin (RR) policy has not been seriously considered for being used in the field of real-time computing. Traditionally, the RR policy was considered useful for low priority processes performing some background computation tasks "when nothing more important is running" (see [1] pp 163). However it has been shown in [12] that RR should not be ruled out a priori because there are cases where this policy is an efficient choice in terms of schedulability as well as for fine tuning the system (e.g. minimizing the end-of-execution jitter for instance in automatic control applications). In addition, RR is part of the Posix1003.1b standard [7] and thus it is implemented in the wide majority of real-time operating systems. The problem addressed here is to minimize the energy consumption of a set of jobs scheduled under the RR policy while meeting the deadline constraints.

Existing work. The two most common techniques for reducing power in processors are Dynamic Power Management (DPM) and Voltage Scaling (VS). DPM compatible processors must possess at least one sleep mode and the whole problem is to decide when to shut down the CPU if it is inactive. The principle of Voltage Scaling is to change the voltage supply and thus the clock frequency of the processor[1]. Two types of voltage scaling can be distinguished. In Static Voltage Scaling (SVS) a single voltage is applied to the system. The voltage is set off-line, for instance through the BIOS of the computer, and it

[1]For each voltage supply there is an optimal frequency which is the highest frequency achievable with the voltage value. In the rest of the document, we will use interchangeably either voltage, frequency or speed of the processor.

will not be modified during the system execution. With Dynamic Voltage Scaling (DVS) it is possible to vary the clock frequency at run-time. It is in general the most effective technique even with the speed switching overheads, but it cannot be used on most systems since the processor and operating systems must be compliant with this technique.

Many papers have proposed power conscious versions of widely known real-time scheduling policies such as Fixed-Priority Preemptive (FPP) or Earliest Deadline First (EDF). Regarding FPP, Shin and Choi describe in [15] a run-time mechanism that takes advantage of slack time to reduce energy. When there is just one active task, frequency is lowered in such a way as to finish the task at the arrival date of the next task. In the case where there is no active tasks, the processor may be shutted down. An off-line algorithm is presented in [16], where the authors propose an algorithm to compute the minimum constant speed applicable to a periodic task set under FPP. In [5], Gruian takes a step further and describes how to obtain the minimum constant speed for each task of the task set.

Quan and Hu in [13] describe an optimal algorithm for voltage scheduling a set of jobs with FPP. Their analysis is based on an EDF schedule transformation but it is restricted to task sets with certain characteristics. More recently, [19] proves that computing the voltage schedule of a set of tasks under FPP is NP-Hard and they present an approximation scheme that runs in polynomial time and whose precision w.r.t. the optimal solution can be chosen arbitrarily small.

When the scheduling is made on top of EDF, Yao et al. in [18] proposed an off-line algorithm for finding the optimal voltage schedule of a set of independent tasks. Their algorithm works by identifying the time interval, termed the critical interval, over which the highest processing speed is required. The lowest admissible frequency is computed for this interval, the tasks belonging to this interval (i.e. arrival date and deadline inside the interval) are then removed and a sub-problem is constructed with the remaining tasks. The complexity of an implementation is $\Theta\left(n^3\right)$. Recently, this scheduling problem has been solved by transforming it in a shortest path problem [3; 4; 2]. In this way, the optimal off-line voltage schedule can be determined with a lower complexity, decreasing it up to $\Theta(n \log n)$ when tasks are FIFO. The case where the number of speeds is finite is also considered and an algorithm minimizing the number of speed changes is provided. For the non-preemptive case, still under EDF, Hong *et al* present an heuristic [6] based on the work of Yao *et al*.

Goal of the study. The objective is to provide an algorithm for finding the minimal processor frequency that still permits a job set with real-time constraints to be successfully scheduled under Round-Robin. The algorithm is based on a schedulability analysis of a job set under RR which is another contribution of the paper. On-line mechanisms for further reducing the energy consumption and ensuring the feasibility when jobs do not require their Worst-Case Execution Time (WCET) are also proposed. Finally, a first solution to the problem when the frequency might be changed on-line is given.

Organization of the paper. The model of the system is described in section 2 which includes the power and task model as well as a formal definition of the Round-Robin policy. Section 3 will be devoted to the schedulability analysis of a set of jobs under RR. In section 4 one focus on the energy reduction problem, featuring off- and on-line proposals. The conclusion of this study is presented in section 5.

2 Model of the system

In this section we introduce the task and the power model used and a formal definition of the RR scheduling policy is given.

2.1 Power model

We are considering an uniprocessor system equipped with two power reduction mechanisms that are DPM and VS. Regarding DPM, one assume two modes: active and sleep. For VS, we are assuming a finite number of processor frequencies. Since a continuous number of processor speeds is not feasible with today's technology [5], this assumption is very reasonable.

As power consumption differences among clock cycles are statistically insignificant (refer to [14]), we assume that all clock cycles require the same amount of energy. The term processor speed is the relative value of the clock frequency f, compared to the maximum clock frequency f_{max}:

$$s_f = \frac{f}{f_{max}}.$$

For example, running at half of the maximum speed $(S_f = 0.5)$ will take twice the time to complete the job. The set SF is the set of all speeds available for the processor.

2.2 Task model

A job is a sequential process that takes a certain time to be executed depending on the amount of clock cycles needed and on the processor frequency. The system under study consists in a set of N independent jobs, $J = \{J_1, J_2, \ldots, J_N\}$ that do not recur over time. Nevertheless, as it is classicaly done for EDF [18; 3] and FPP [19], it is possible to accommodate the system to periodic tasks by considering all instances until the *LCM* of the tasks period (or $2 \cdot LCM$ plus the maximum of the response times when the first instance of all tasks are are not released simultaneously). Job J_i is characterized by a tuple of the form $\langle A_i, C_i, \psi_i, D_i \rangle$. Let us define these timing parameters for a job J_i:

- A_i: arrival time of the job. At time A_i, J_i is ready to execute.
- C_i: worst-case execution time at maximum frequency.
- ψ_i: it is the length of the quantum of J_i under RR.
- D_i : deadline of job J_i. Hard real-time constraints are imposed in our model thus missing deadlines cannot be tolerated as it can jeopardize the system.

Associated with each job J_i, we denote E_i as the time instant on which job J_i fully finishes its execution which clearly depends on the scheduling policy used. The set $\{E_i\}$ denotes the set of end of execution times while $\{D_i\}$ is the set of deadlines of all jobs J_i from set J with $1 \leq i \leq N$.

An RR-Cycle is defined as a time interval that starts by the execution of the lowest index job that has pending work and finishes when all active jobs have used the processor once for at most its quantum time. The sum of all quantums of the jobs with index lower or equal to i is $\Upsilon_i = \sum_{j \leq i} \psi_j$.

2.3 The Round-Robin policy

At each instant, a scheduling policy chooses among the set of all active instances exactly one instance for being executed on the processing unit. The idea presented in [10] is to define scheduling policies through a priority function $\Gamma_{k,n}(t)$ that gives the priority of every instance $\tau_{k,n}$ at any time t. The resource allocation rule is the *Highest Priority First Rule* which means that the scheduler will always select the pending job with the highest priority for being executed.

The function $\Gamma_{k,n}(t)$ takes its values from a totally ordered set which can been chosen as the set of multidimensional $\mathbb{R}$-valued vectors $\mathcal{P} = \{(p_1, \ldots, p_n) \in \mathbb{R}^n \mid n \in \mathbb{N}\}$. As no natural order exists for vectors, a lexicographic order where the i^{th} component is taken as the i^{th} "letter" of a word consisting of real numbers is defined. Vectors are compared component per component and the convention is that smaller the numerical value, bigger the priority. For instance $(1,2,3) \succ (1,2,4)$ where $\succ$ means "strictly bigger".

A priority function for RR has been proposed in the context of recurrent tasks [11], the priority function for jobs is basically a simplification of the one proposed in [11]. The priority of a job J_k at time t is:

$$\Gamma_k(t) = \left(\left\lfloor \frac{\int_{A_k}^{t} \prod_k(x)dx}{\psi_k} \right\rfloor + P(A_k),\ k \right) \tag{1}$$

where:

- $P(t)$ is the number of RR-Cycles that have been completed from U_k until t.
- $W_k(t)$ is the amount work that remains to be done for job J_k at time t.
- $W_{1\ldots N}(t) = \sum_{k=1}^{N} W_k(t)$, the total amount of work to be done for the whole job set.
- $U_k = \max\{t \leq A_k : W_{1\ldots N}(t) = 0\}$. This is the first date before or at A_k such that there is no pending work, it is usually termed the beginning of an interference period in the literature.
- $\prod_k(t) = \begin{cases} 1, & \text{if } J_k \text{ uses the processor at time } t \\ 0, & \text{if } J_k \text{ doesn't uses the processor at time } t \end{cases}$

The first term of the first component of the vector, see equation (1), guarantees that a job uses the processor during its whole quantum time if enough work is pending which is the basic functioning scheme of RR. When a new job is activated, the second element $P(A_k)$ prevent the job from monopolizing the processor due to its absence in the previous RR-Cycles. The last component of the vector ensure that two jobs will not share the same priority if their first components are equal. Note that only the jobs that are active at time t (*i.e.* $J_i s.t. A_i \leq t$) can be chosen to be executed whatever their priority.

3 Feasibility under Round-Robin

In this section, we propose a feasibility analysis of a set of jobs under Round-Robin which will be used for the frequency reduction strategy described in section 4. This schedulability analysis is not a feasibility test but an explicit computation of response times that have to be compared to deadlines for assessing feasibility.

3.1 Algorithmic details

The basic idea behind the algorithm is that it is possible to determine the work done for each job during a certain time interval by counting the number of elapsed RR-Cycles. A new "time period" starts whenever the length of the RR-Cycle time changes due to the activation of at least a job. The RR-cycle also changes when a job leaves the system but that will be taken into account in a second step. For evaluating the work done for each job, and thus deriving the execution end, one jumps from one time period to another. If at the end of a time period the CPU time allocated to a job is greater or equal to its WCET then the job has been finished during this time period and a closer examination of the exact end of execution date has to be performed (using function DetermineEnd labeled as algorithm 2).

3.1.1 Determining time periods

The complete algorithm is divided into several parts. The first step is to determine the ordered set of the successive "time periods" denoted as K. It is defined in a recursive way :

$$\begin{cases} K_1 = A_1. \\ K_u = K_{u-1} + \left\lfloor \frac{A_u - K_{u-1}}{\Upsilon_{u-1}} \right\rfloor \cdot \Upsilon_{u-1} \quad , \text{if } u > 1. \end{cases} \tag{2}$$

This set is constituted by all time instants at which a new RR-cycle starts with at least one additional job. If we have any $K_a = K_b$, with $1 \leq a \leq N$ and $1 \leq b \leq N$, then one just include $K_{min(a,b)}$ into K. This is the case when there is more than one new active job that arrives in the same RR-cycle. Consider $succ(K_i)$ as the element from set K that immediately succeeds K_i. For the element K_N, $succ(K_N) = \infty$.

The execution thread is initiated in the RRFT function (see algorithm 1) by determining the first two elements of set K, which allows to compute the work performed for all active jobs during the time period (a time period corresponds to the interval time between K_u and $succ(K_u)$). The set J' is formed by all

jobs that have finished their execution during the current time period and their precise end of execution dates are determined in function DetermineEnd (see algorithm 2). Then the job is updated in the following way (end of algorithm 1):

- if no job has finished in the current time period, the remaining work is adjusted here. Otherwise the work is adjusted at step 3 of DetermineEnd since it may happen that the RR-cycle finishes before the predicted date and thus that job might not execute at all in the RR-cycle (see figure 1 and explanations at the end of §3.1.2).
- the activation dates before $succ(K_1)$ are shifted to $succ(K_1)$.

The "While" loop of algorithm 1 is executed until all jobs are finished.

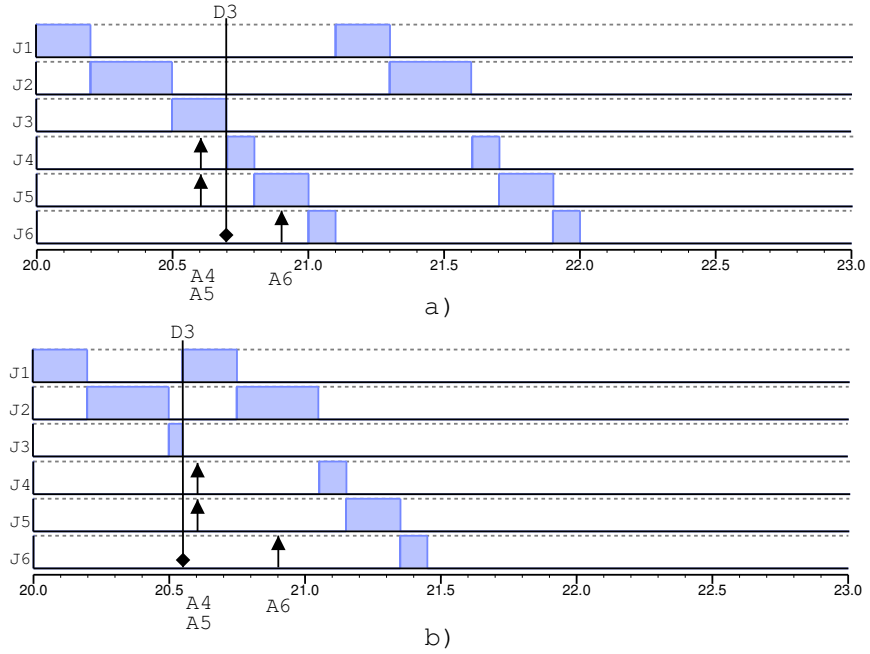

Figure 1: Scheduling special cases after job termination. On figure b), the end of execution of J_3 induces a new time period and thus J_4, J_5 and J_6 are only executed in the first RR-cycle of the next time period. Figure a) shows the case where J_3 does not finish before the end of its quantum.

Algorithm 1 RRFT

Input: Job set $J = \{J_i(A_i, C_i, \Psi_i)\}$ ordered by arrival dates.
Output: Index ordered vector of $\{E_i\}$ from job set J

While $J \neq \emptyset$
  Determine K_1 and $succ(K_1)$ of K from job set J. /* Set K defined in section 3.1 */
  For each J_i active between K_1 and $succ(K_1)$,
    If $W_i(succ(K_1)) = 0$ **Then** /* $W_i(succ(K_u)) = \left(W_i(K_u) - \frac{succ(K_u) - K_u}{\Upsilon_i} \cdot \Psi_i\right)^+$ */
      Add $J_i'(A_i, C_i, \Psi_i)$ in J'
    End If.
  End For.
  If $J' \neq \emptyset$ **Then**
    Run DetermineEnd(J, J', K_1) /*Defined in Algorithm 2.*/
  Else
    Update remaining work of jobs at point $succ(K_1)$.
    Update activation date: $A_i \leq succ(K_1) \Rightarrow A_i = succ(K_1)$
  EndIf.
End While.
End.

3.1.2 Determining execution ends

The DetermineEnd function (refer to algorithm 2) determines the ends of execution during the current time period (whose beginning is denoted K_u). The function starts by determining the first job that finishes its execution between K_u and $succ(K_u)$ (step 1 in algorithm 2). Then, at step 2, one identifies the jobs starting their execution in the next RR-cycle due to the first end of execution. In this case, the beginning of the next RR is also the start of a new time period. For instance, in figure 1b), J_4 will be executed in the next RR-cycle contrarily to what was planned.

At step 3, an update of the remaining jobs characteristics is carried out. The value of the smallest element of set $\mathcal{A}$ allows to distinguish between the jobs that executed some work in the time period and the jobs that did not. In the latter case, the only required modification will be the value of the job index due to the termination of at least one higher priority job (the set F_i in algorithm 2 is formed by all finished jobs with index lower than i).

Algorithm 2 DetermineEnd

Input: Job set $J = \{J_i(A_i, C_i, \psi_i)\}$, job set $J' = \{J'_i(A'_i, C'_i, \psi'_i)\}$, value K_u from K set.
Output: Determines the end of execution time of job J_d (at least).

Step 1: Determine the first job that finishes execution in period K_u until $succ(K_u)$.
/* m is the number of used quantums, d is the index of the job. */
$(m,d) = \min\left\{(m',d') \mid m' = \left\lceil \frac{W_{d'}(K_u)}{\psi_{d'}} \right\rceil, \forall d' = 1...u-1, m' \in \mathbb{N}^+, J'_d \in J'\right\}$.
/* h is the index of the highest job active between K_u and $succ(K_u)$. */
$h = \max\{i \mid A_i \leq succ(K_u) \wedge W_i(K_u) > 0,\ J_i \in J\}$
$E_d = K_u + (m-1)\cdot\Upsilon_h + \Upsilon_{d-1} + W_d(K_u) - (m-1)\cdot\psi_d$.
Output E_d from job J_d .
If $J \setminus J_d = \emptyset$ **Then**
End.
End If

Step 2: Determine the start of the following RR-Cycle.
/* $\mathcal{A}$ is the index set of jobs that will start execution in the next RR-Cycle. */
Consider $\mathcal{A} = \left\{ i \mid A_i \geq E_d + \sum_{d<b<i} \min\{\psi_b, W_b(E_d)\} \wedge i \leq h, \quad J_i \in J \right\}$.

Step 3: Update remaining jobs' characteristics.
Consider $F_i = \{\text{jobs finished in the current } K \text{ period with index inferior to } i\}$.
Let $p = \min(\mathcal{A})$.
Each J'_i that composes the job set J' is re-defined in the following manner:

$$J'_i = \begin{cases} W'_i = \begin{cases} W_i(K_u) - m\cdot\psi_i, & i < d \\ W_i(K_u) - (m-1)\cdot\psi_i - \min\{W_i(K_u), \psi_i\}, & i > d \end{cases} \\ \psi'_i = \psi_i \\ A'_i = \begin{cases} \max\{E_j \mid T_j \in F_p\}, & i < p \\ A_i, & i \geq p \end{cases} \\ i = \begin{cases} i, & i < d \\ i - \sharp F_i, & i > d \end{cases} \end{cases} \text{, for } i = 1...p-1$$

Remove finished jobs from J', with their ends of execution determined.
$J'_i = J_{i+\sharp F_p}$, for $i = p...N$.
$J = J'$.

Regarding the jobs that executed some work, besides the priority index (same strategy as above), one has to modify the arrival time and the remaining work. The computation for finding the exact remaining work for each job at the end of the time period is then performed. One denotes d the index of the job that finishes first its execution in the RR-cycle. Lower index jobs use the same number of quantums (case $i < d$ in the update of W'_i). Higher index jobs, however, might not completely use their last quantum time (used after the first job has finished), since they might end their execution before. Thus, a comparison between their remaining time and their quantum size has to be performed (case $i > d$ in the update of W'_i). Then the arrival time of each active job is shifted to the last end of execution date of the RR-cycle. These new arrival dates will be used to compute the next K set elements at the

beginning of the While loop in function RRFT (see algorithm 1). At the end of DetermineEnd, at least one job has finished its execution, the remaining jobs attributes were updated (index, remaining execution time, arrival date) and the set of jobs J is updated accordingly.

3.2 Complexity and performance

During the algorithm at most $2N$ time periods are created: at most N induced by an arrival and N by an end of execution. For each time period, one needs to update the job's characteristics which is performed at most in $4N$ time in the DetermineEnd algorithm. The overall complexity is polynomial in $\Theta(N^2)$.

The correctness of the algorithm has been validated on several hundred of experiments against a naive feasibility test that consists in simulating the scheduling. Logically, our proposal proves to be faster than the naive implementation when the number of quantums used by each task becomes large (greater than 15 with our implementation).

4 Energy reduction

In this section, we provide energy reduction techniques for scheduling under RR on an uniprocessor system. We first consider the problem of finding a single speed that ensures the feasibility during the whole lifetime of the system based on the WCET. Then, since WCET of a job is hardly reached, we propose on-line mechanisms for further reducing the consumption through DPM and ensuring schedulability when jobs consume less than their WCET. Finally, one describes a first non-optimal extension to the case where several speeds might be used on-line.

4.1 Off-line analysis

The following observation allows to reduce the number of feasible processor speeds which will narrow the search space for this voltage scheduling problem.

Observation 1. The single optimal processor speed under RR is higher or equal to the maximum of the speeds ensuring feasibility under EDF.

Proof: A direct consequence of Theorem 1 in [18] is that the minimum speed needed to ensure feasibility under EDF, denoted S_{edf}, is the the utilization factor of the "critical interval" of the job set. The utilization factor $u(t_1,t_2)$ of an

interval $[t_1,t_2[$ is intuitively the quantity of CPU needed to successfully schedule under EDF jobs belonging to this interval (arrival and deadline inside the interval). The critical interval is the time interval that maximizes the utilization factor, thus:

$$S_{edf} = \max_{0 \leq t_1 \leq t_2} u(t_1,t_2) = \frac{S_J(t_1,t_2)}{t_2 - t_1} \tag{3}$$

where $S_J(t_1,t_2)$ is the workload of jobs of job set J with deadlines lower than t_2 and arrival in $[t_1,t_2[$. Since EDF is optimal for the scheduling of independent jobs (see [8] quoted in [17]) no other policy can successfully schedule the jobs belonging to this interval at a lower speed than EDF. The speed under RR is thus greater or equal to S_{edf}. □

From now on, the only speeds considered are the ones that remain inside the interval that begins in S_{edf} (observation 1) and ends in $\max\{SF\}$. Using the feasibility test of section 3, one can test for each speed of $SF' = \{s_i \,|\, s_i \in SF, S_{edf} \leq s_i\}$ the schedulability under RR. The WCETs taken for the jobs have to be updated w.r.t the frequency for which the feasibility is tested; at speed s_f, the time needed for executing job J_i is C_i/s_f where C_i is the WCET at the maximum frequency (see paragraph 2.2). Intuitively, we might envisage that the search through the set SF' could be directed using a classical binary search procedure which would ensure at most $\log_2 \#SF'$ feasibility tests (cf. [9]) but it is actually not possible as shown in Observation 2.

Observation 2. Under Round-Robin, feasibility is not monotonous in the frequency of the CPU.

Proof: We will show on a counter example that a job set might be feasible with speed S_1 and not feasible with $S_2 > S_1$. Let us consider the job set defined in table 1. The scheduling of this job set is shown on figure 2 at speed 1 (sub-figure a) and speed 0.8 (sub-figure b). One sees that job J_4 does not meet its deadline constraint at speed 1 while it is schedulable at speed 0.8 . □

The search for the minimum acceptable speed has to start from speed $\min\{s_i \,|\, s_i \in SF, s_i \geq S_{edf}\}$ and it stops at $\max\{SF\}$. In the worst-case, every single speed in SF' has to be tested to assess the feasibility of the system. Observation 2 also implies that even without any interests in lowering energy consumption, it might be useful to consider lower frequency for the schedulability of the system.

	A_i	C_i	$C_i^{0.8}$	ψ_i	D_i
J_1	0	16	20	8	45
J_2	5	16	20	8	50
J_3	34	32	40	16	90
J_4	52	4	5	5	64

Table 1: Characteristics of figure 2 job set. C_i denotes the WCET of job i at speed 1 while $C_i^{0.8}$ is the WCET at speed 0.8 .

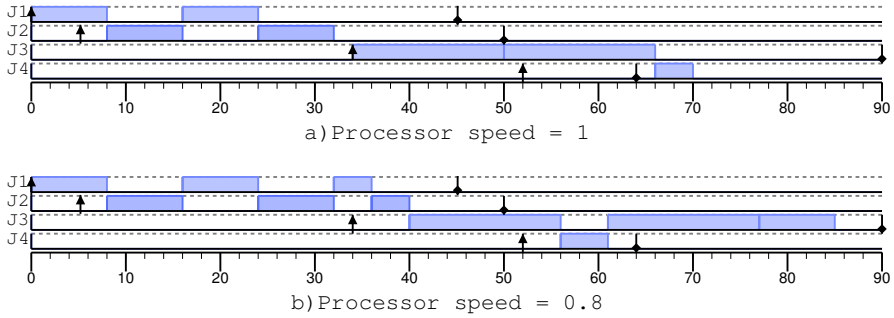

Figure 2: Feasibility under different processor speeds. In figure a), the job set scheduled at maximum speed is not feasible. Surprisingly, lowering the speed to 0.8 enable the job set to be feasible.

4.2 On-line mechanisms

At run-time, due to a better knowledge of the job set that is being scheduled, some improvements can be made in order to further lower the energy gains provided by off-line analysis. This better knowledge comes from the fact that it is known when a job finishes its execution. Whenever the execution time is shorter than the job's WCET, the spare time left can be used for more energy savings. However, feasibility has to be ensured.

Observation 3. The termination of a job earlier than its expected WCET may provoke the unfeasibility of the job set under Round-Robin.

Intuitively, one could say that an earlier termination of a job will not provoke other jobs to miss their deadline. However, under RR this doesn't remain true. As an example, refer to figure 3. In figure 3a), all jobs need their WCET while in figure 3b), job J_2 terminates before its WCET (at time 32). Due to

the earlier termination of job J_2, an extra RR-cycle is being scheduled before the first quantum of job J_5. And as seen in the example, it will turn the job set unfeasible since J_5 finishes after its deadline.

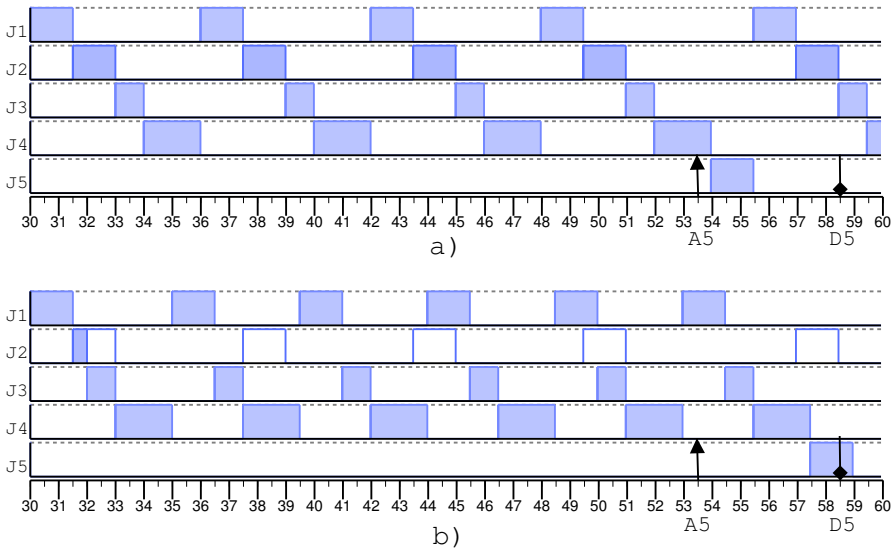

Figure 3: Unfeasibility of the job set due to earlier completion of J_2. Figure b) corresponds to the case where J_2 finishes earlier than planned, one sees that job J_5, arrived at time 53.5, does not meet its deadline set to 58.5 .

In the following, we propose two strategies for ensuring feasibility and reducing consumption. Both require the knowledge of the estimated end of execution date, denoted E_i for job J_i, that is computed off-line when assessing feasibility.

Quantum Sleep Method. The time allocated to the job and not used due to the early completion is simply spent on waiting. Two solutions are possible, either the processor is shutted down in sleep mode or NOP operations are executed (NOP only consume 20% of the average energy taken by other instructions, see [15]). The first solution might not be feasible due to the switching mode overhead. If both techniques are possible, since the threshold at which the sleep mode becomes more effective can be easily computed off-line, the strategy leading to the most energy savings is selected at run-time.

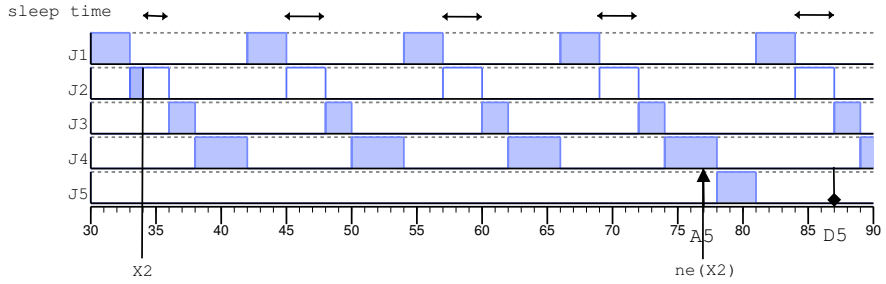

Figure 4: Quantum Sleep Method.

Group Quantum Sleep Method. This is an optimization w.r.t. the previous solution that consists in grouping sleep instants whenever possible. It reduces the amount of switches between processor modes and, for DPM, it might enable deeper sleep mode. However, special concern must be taken into account regarding Observation 3.

After the earlier end of execution of a job, several particular "events" can follow such as the arrival of a job or the planned end of execution time of the job that finishes earlier. When the next event after an earlier completion is the planned end of execution date of the job, we propose a method to group the unused quantums while ensuring the schedulability of the remaining jobs after the sleep period.

The amount of time that can be spent in sleep mode due to the termination of job J_i at time t before the next event $ne_i(t)$ is

$$SLEEP(J_i,t) = Qa(t) + Qb(t) + S_i(t+Qa, ne_i(t) - Qb) \qquad (4)$$

where:

- X_i is the actual end of execution of J_i,
- C_i^* is the actual execution time of J_i ,
- $S_i(t_1,t_2)$ is the amount of time that was planned to be allocated to job J_i between t_1 and t_2,
- $ne_i(t)$ is the next "event" after time t for job J_i,
- $1_{[a]} = \begin{cases} 1, & \text{if the predicate } a \text{ is true} \\ 0, & \text{otherwise} \end{cases}$

The total amount of time that can be clustered is an addition of three parts as seen in equation 4. The portion of quantum remaining after J_i's end of execution at time X_i is calculated as $Qa(t) = (\Psi_i - (C_i^* mod \Psi_i)) \cdot 1_{[t=X_i]}$(first term of 4). The portion of the last quantum where J_i was planned to finish corresponds to $Qb(t) = (C_i \ mod \ \psi_i) \cdot 1_{[ne_i(t)=E_i]}$(second term of 4). The time period of full-sized quantums of J_i located between the first and last quantum corresponds to $S_i(t+Qa, ne_i(t) - Qb)$ (third term of 4).

The sleep period starting point will be $ne_i(t) - SLEEP(J_i,t)$ with a duration of $SLEEP(J_i,t)$. Until time $ne_i(t) - SLEEP(J_i,t)$ all jobs except the one that finishes earlier are scheduled without changes. Figure 5a) illustrates this strategy. The feasibility is preserved since $\forall x \in [t, ne_i(t) - SLEEP(J_i,t))$, $S_j(t,x) \geq S_j^{'}(t,x)$ where $S_j^{'}(t_1,t_2)$ is the amount of CPU allocated to job J_j between t_1 and t_2 under the scheduling with the sleep interval. The scheduling after $ne_i(t)$ remains unchanged.

Our clustering strategy also applies when the next event after the early completion of job J_i is the beginning of a sleep period due the completion of a job J_j. It is the case for J_1 in figure 5b). Considering this possibility, the next event for job J_i finishing at time t is $ne_i(t) = \min\{E_i \cup (ne_j(X_j) - SLEEP(J_j,X_j))\}$.

After a sleep period, there still might exist unused quantums belonging to a job that has finished before the sleep period (e.g. job J_1 in figure 5.b)). In this case, one can iteratively use equation 4 to create several sleep periods in the timeline. For computing the following clusters of unused quantums, the parameter t in equation 4 would not be the actual end of execution time but the end of the sleep period before the first quantum one wants to group ($ne(X_2)$ in figure 5.b)).

4.3 Extensions

Considering just one processor speed for the whole job set is clearly not an optimal solution when the processor and the operating system enable the speed to be changed at run-time. Indeed, with a single speed, the most constrained job dictates its speed requirement to all jobs during the whole lifetime of the system.

A first solution to the case where multiple frequencies are possible is to determine "time intervals" such that the choice made for the frequency during one time interval cannot interfere with the scheduling inside other intervals. This is possible if time intervals are all separated by processor idle times. The time intervals can be chosen as the largest possible processors busy periods corresponding to the case where all jobs execute until their deadlines. The

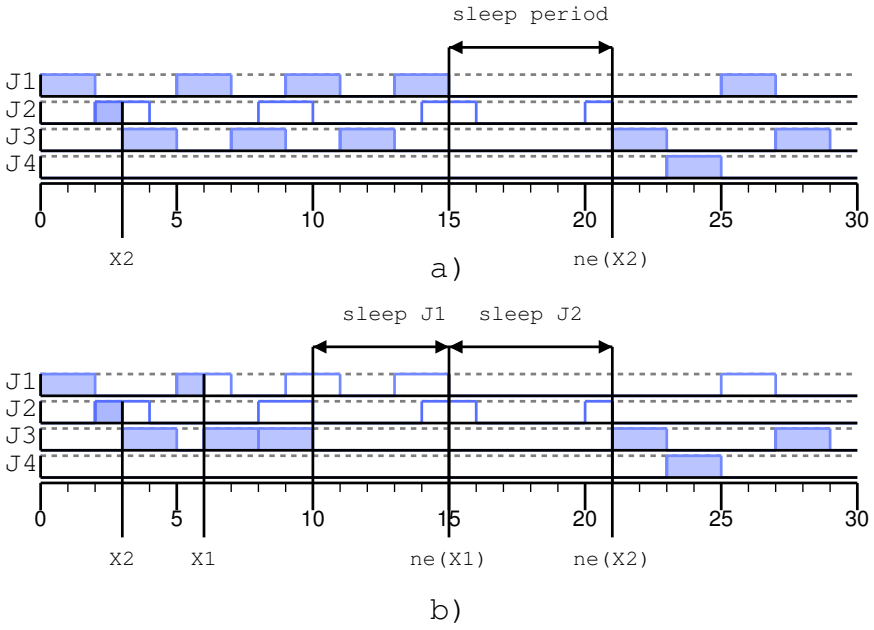

Figure 5: Group Quantum Sleep Method. In a), after the end of execution of J_2 at point X_2, the point $ne(X_2)$ and the value $SLEEP(J_2, X_2)$ are calculated, defining time 15 as the instant where the processor enters sleep mode. In b), the end of execution of J_1 increases the sleep period due to J_2, shifting its start to 10.

remaining idle times cannot be used whatever the scheduling decisions and thus the time intervals between idle times cannot interfere one with each other. The beginning of interval i is denoted $\mathcal{A}_i$ and its end $\mathcal{E}_i$, they can be computed iteratively :

$$\mathcal{A}_i = \min\{A_i \geq \mathcal{E}_{i-1}\} \qquad \text{and}$$

$$\mathcal{E}_i \quad = \quad \min\{D_j > \mathcal{A}_i \,|\, \nexists A_k < D_j \,\text{with}\, D_k > D_j,\ j = 1..N\}$$

with $\mathcal{E}_0 = \min\{A_i\}$. Note that the computation of $\{\mathcal{A}_i\}$ and $\{E_i\}$ can be carried out during the feasibility test without adding significant overheads.

If the low power algorithm from section 4.1 is applied separately to each of the time intervals, it can only reduce energy consumption since the highest speed of the groups will be local to the group it belongs to and it will not be the speed chosen for all time intervals.

5 Conclusion

In this paper, we address the problem of scheduling a set of real-time job under Round-Robin with the objective of minimizing the energy consumption while meeting feasibility constraints. We first considered the problem of finding a single speed that ensures the feasibility during the whole lifetime of the system based on the WCET. Our solution is optimal in the context of a single frequency as the use of a lower frequency will not ensure feasibility. Since the WCET of a job is not always reached, we proposed on-line mechanisms that use the gain time for further reducing the consumption through DPM and for ensuring schedulability. Another contribution of the paper is an efficient feasibility test for Round-Robin which, to our best knowledge, has not been proposed in the context of non recurrent tasks.

A first counter-intuitive result is that the end of execution of a job before its WCET might lead to a non-feasible schedule. It has also been highlighted that a job set might not be feasible at maximum frequency while being feasible with a lower frequency. This implies that even without interests in energy saving, lower frequencies have to be considered for feasibility.

We extend the analysis to the case where several frequencies might be used on-line and propose a first solution to this problem by applying the single frequency solution on time intervals that cannot interfere one with each other. More efficient approaches may be perhaps derived from a fluid approximation of the Round-Robin policy.

References

[1] B. Gallmeister. *POSIX.4: Programming for the Real World*. O'Reilly & Associates, January 1995.

[2] B. Gaujal and N. Navet. A new EDF feasibility test. Technical report, to appear, INRIA, 2004.

[3] B. Gaujal, N. Navet, and C. Walsh. Real-time scheduling for optimal energy use. In *Faible Tension Faible Consommation (FTFC'2003)*, pages 159–166, 2003.

[4] B. Gaujal, N. Navet, and C. Walsh. Real-time scheduling for optimal energy use. Technical report, INRIA N°4886, 2003. In submission.

[5] F. Gruian. *Energy-Centric Scheduling for Real-Time Systems*. PhD thesis, Lund Institute of Technology, 2002.

[6] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. Srivastava. Power optimization of variable voltage core-based systems. In *Design Automation Conference*, pages 176–181, 1998.

[7] (ISO/IEC). *9945-1:1996 (ISO/IEC)[IEEE/ANSI Std 1003.1 1996 Edition] Information Technology - Portable Operating System Interface (POSIX) - Part 1 : System Application : Program Interface*. IEEE Standards Press, 1996. ISBN 1-55937-573-6.

[8] J.R. Jackson. Scheduling a production line to minimize maximum tardiness. Technical report, University of California, 1955. Report 43.

[9] D.E. Knuth. *The Art Of Computer Programming, Volume 3: Sorting and Searching*. Addison Wesley, second edition, 1998.

[10] J. Migge. *Scheduling of Recurrent Tasks in One Processor: a Trajectory Based Model*. PhD thesis, Nice University, 1999.

[11] J. Migge, Alain Jean-Marie, and N. Navet. Timing analysis of compound scheduling policies : Application to Posix1003.1b. *Journal of Scheduling*, 6(5):457–482, 2003.

[12] N. Navet and J. Migge. Fine tuning the scheduling of tasks through a genetic algorithm: Application to Posix1003.1b compliant systems. *IEE Proceedings - Software*, 150(1), 2003.

[13] G. Quan and X. Hu. Minimum energy fixed-priority scheduling for variable voltage processors. In *Design Automation and Test in Europe*, 2002.

[14] J. Russell and M. Jacome. Software power estimation and optimization for high performance, 32-bit embedded processors. In *International Conference on Computer Design (ICCD'98)*, 1998.

[15] Y. Shin and K. Choi. Power conscious fixed priority scheduling for hard real-time systems. In *Design Automation Conference*, pages 134–139, 1999.

[16] Y. Shin, K. Choi, and T. Sakurai. Power optimization of real-time embedded systems on variable speed processors. In *International Conference on Computer-Aided Design*, pages 365–368, 2000.

[17] J.A. Stankovic, M. Spuri, K. Ramamritham, and G.C. Buttazo. *Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms*. Kluwer Academic Publisher, 1998.

[18] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *lEEE Annual Foundations of Computer Science*, pages 374–382, 1995.

[19] H.-S. Yun and J. Kim. On energy-optimal voltage scheduling for fixed-priority hard real-time systems. *ACM Transactions on Embedded Computing Systems*, 2(3):393–430, August 2003.