Schedule_An_Instance (τ_i^j):
Input: A linked list, a task instance τ_i^j and a sequence of sorted free intervals, $S = \{ int_1, int_2, \ldots, int_{|S|} \}$,
in which each interval overlaps $[est(\tau_i^j), lst(\tau_i^j)]$.

Let the execution time of τ_i^j be e.
For $n = 1$ to $|S|$ do
Let int_n be $[\ell,\mu]$.
If $\mu - \ell \geq e$ then
Return a new time slot with *start time* $= \ell$ and *finish time* $= \ell + e$.
End if.
End for.
Compute *left laxity* and *right laxity* for each time slot in the linked list by equations 22 and 23.
For $n = 1$ to $|S|$ do
Let int_n be $[\ell,\mu]$.
If $\ell \geq s_i^{j-1} + p_i$ then /* Try left shift first then right shift */
Let the time slot that immediately precedes int_n be w_k.
If *left_laxity*$(w_k) + \mu - \ell \geq e$ then /* Left shift */
Left_Shift$(w_k, e - \mu + \ell)$.
Return a new time slot with *start time* $= \mu - e$ and *finish time* $= \mu$.
Else
Let the time slot that immediately follows int_n be w_k.
If *right_laxity*$(w_k) + \mu - \ell \geq e$ then /* Right shift */
Right_Shift$(w_k, e - \mu + \ell)$.
Return a new time slot with *start time* $= \ell$ and *finish time* $= \ell + e$.
End If.
End If.
Else /* Try right shift first then left shift */
Let the time slot that immediately follows int_n be w_k.
If *right_laxity*$(w_k) + \mu - \ell \geq e$ then /* Right shift */
Right_Shift$(w_k, e - \mu + \ell)$.
Return a new time slot with *start time* $= \ell$ and *finish time* $= \ell + e$.
Else
Let the time slot that immediately precedes int_n be w_k.
If *left_laxity*$(w_k) + \mu - \ell \geq e$ then /* Left shift */
Left_Shift$(w_k, e - \mu + \ell)$.
Return a new time slot with *start time* $= \mu - e$ and *finish time* $= \mu$.
End If.
End If.
End If.
End for.
Schedule τ_i^j at the end of linked list.

Table 3: The Scheduling of a Task Instance

References

[1] T. Carpenter, K. Driscoll, K. Hoyme, and J. Carciofini. Arinc 659 scheduling: Problem definition. In *Proceedings of IEEE Real-Time Systems Symposium*, San Juan, PR, Dec. 1994.

[2] M. L. Dertouzos and A.K. Mok. Multiprocessor on-line scheduling of hard real-time tasks. *IEEE Transactions on Software Engineering*, 15(12):1497–1506, Dec. 1989.

[3] M.G. Harbour, M.H. Klein, and J.P. Lehoczky. Fixed priority scheduling of periodic tasks with varying execution priority. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 116–128, Dec. 1991.

[4] Krithi Ramamritham. Allocation and scheduling of complex periodic tasks. In *Proceedings of the 10th International Conference on Distributed Computing Systems*, pages 108–115, Paris, France, 1990.

[5] T. Shepard and J.A.M. Gagne. A pre-run-time scheduling algorithm for hard real-time systems. *IEEE Transactions on Software Engineering*, 17(7):669–677, July 1991.

[6] K. Tindell, A. Burns, and A. Wellings. An extendible approach for analyzing fixed priority hard real-time tasks. *Real-Time Systems*, 6(2), March 1994.

[7] J.P.C. Verhoosel, E.J. Luit, D.K. Hammer, and E. Jansen. A static scheduling algorithm for distributed hard real-time systems. *Real-Time Systems*, pages 227–246, 1991.

[8] J. Xu and D.L. Parnas. Scheduling processes with release times, deadlines, precedence, and exclusion relations. *IEEE Transactions on Software Engineering*, 16(3):360–369, March 1990.

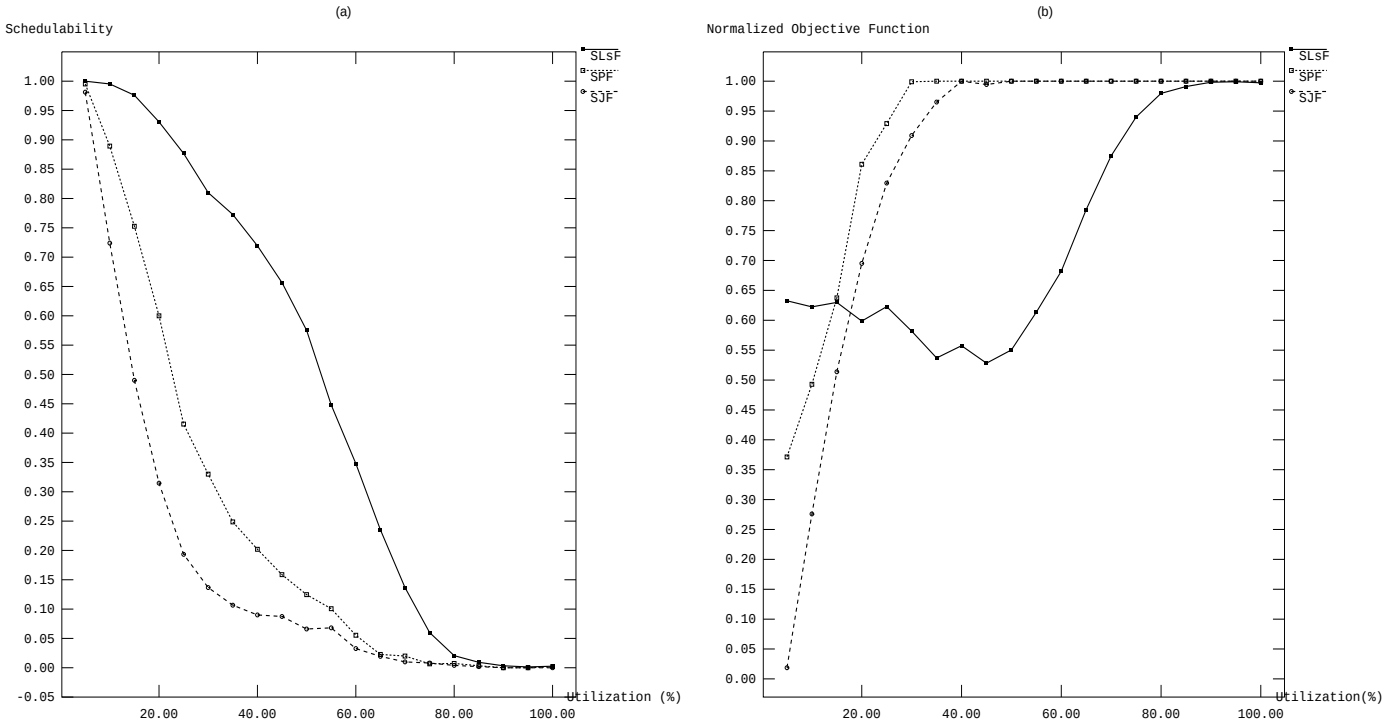

Figure 7: The comparison of three algorithms

high-utilization cases, the schedulability dominates the normalized objective function, and SLsF outperforms the other two algorithms in these regions.

7 Summary

In this paper we have considered the static non-preemptive scheduling algorithm on a single processor for a set of periodic tasks with hybrid timing constraints. The time-based scheduling algorithm is used to schedule a task instance once the scheduling window of the instance is given. We also have presented three priority assignment algorithms for the task instances and conducted experiments to compare the performance. From the experimental results, we see that the SLsF outperforms the other two algorithms.

The techniques presented in this chapter can be applied to multi-processor real-time systems. Communication and synchronization constraints can be also incorporated. In our future work, the extension to a distributed computing systems will be investigated.

- The execution time of a task, e_i : It has the uniform distribution over the range $[0, \frac{p_i}{15}]$, where p_i is the period of the task τ_i. The execution time could be a real value.
- The jitters of a task: $\lambda_i = \eta_i = 0.1 \times p_i + 2 \times e_i$.

The generation scheme for the second experiment is similar to the first one. Given an utilization value U, a set of 20 tasks is randomly generated according to the parameters listed above and then the execution time of each task is normalized in order to make the utilization value equal to U exactly.

We generate 5000 cases of different task sets for each utilization value ranging from 0.05 to 1.00. The schedulability degree of each algorithm on a particular utilization value is obtained by dividing the number of solved cases by 5000. We compare the schedulability degrees of the algorithms on different utilization values. The results are shown in Figure 7(a).

As can be see in Figure 7(a) the SLsF algorithm outperforms the other two algorithms. For example, when the utilization = 50%, the schedulability degree of SLsF is 0.575 while those of SPF and SJF are less than 0.2. It is because the way of assigning the priorities to the task instances in the SLsF algorithm reflects the urgency of task instances by considering the latest start times.

We also compare the objective function value π in Equation 21 among the three algorithms. We define the normalized objective function for an algorithm as

$$\hat{z} = \sum_{i=1}^{5000} \frac{z_i}{5000}, \tag{25}$$

$$\text{where } z_i = \begin{cases} 1 & \text{if the algorithm can not find a feasible solution to case } i. \\ 0 & \text{if } max(i) = min(i). \\ \frac{\pi_i - min(i)}{max(i) - min(i)} & \text{otherwise.} \end{cases}$$

Given case i, the values of $min(i)$ and $max(i)$ are calculated among the objective values obtained from the algorithms which solve the case. For the algorithms which can not find a feasible solution to case i, the objective values are not taken into account when $min(i)$ and $max(i)$ are calculated. The results of the normalized objective functions for each algorithm on different utilization values are shown in Figure 7(b).

It is observed that in the low-utilization cases SJF finds feasible solutions with smaller objective values. It is because that SJF schedules the tasks with the smallest jitters first. By scheduling the tasks with smaller jitter value first it is more easier to make the instances of a task one period apart. we can find a feasible solution with smaller objective value. However, in the middle- or

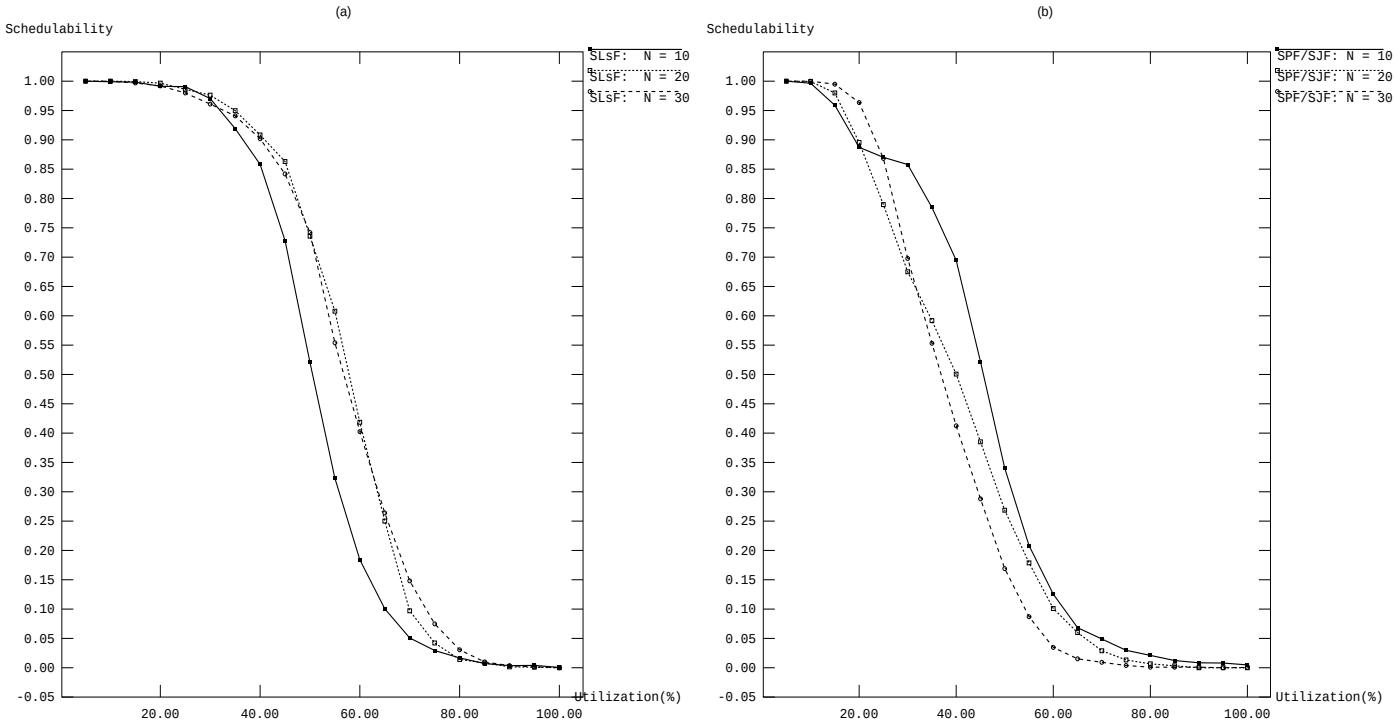

Figure 6: The effect of the numbers of tasks on the schedulability

as the number of tasks in a system becomes bigger. It is beacuse the execution time of a task becomes smaller as the number of tasks increases. For a task system with smaller execution time distribution, the chance for SLsF to find a feasible solution is bigger. The same phenomenon is also found in Figure 6(b) for SPF and SJF in the low-utilization cases (i.e. $U \leq 20\%$). However, for the high-utilization cases in Figure 6(b), the complexity of the number of tasks dominates the algorithms and the schedulability decreases.

6.2 The Second Experiment

The task generation scheme for the second experiment is characterized by the following parameters.

- LCM = 300
- The number of tasks is 20.
- Periods of the tasks: We consider the factors of the LCM as the periods. They are 20, 30, 50, 60, 100, 150, and 300. There may be more than one task with the same period.

the schedulability of each algorithm. The objective of the second experiment is to compare the performance of the three algorithms.

6.1 The First Experiment

The task generation scheme for the first experiment is characterized by the following parameters.

- Periods of the tasks: We consider a homogeneous system in which the period of one task could be either the same as or multiple of the period of another. We consider a system with 40, 80, 160, 320, and 640 as the candidate periods. There may be more than one task with the same period.
- The execution time of a task, e_i : It has the uniform distribution over the range $[0,\frac{p_i}{16}]$, where p_i is the period of the task τ_i. The execution time could be a real value.
- The jitters of a task: $\lambda_i = \eta_i = 0.1 \times p_i$.

We define the utilization of a task system as

$$\sum_{i=1}^{N} \frac{e_i}{p_i} \tag{24}$$

In the first experiment, the utilization value and the number of tasks in a set are the controlled variables. Given an utilization value U and the number of tasks N the scheme first generates a run of raw data by randomly generating a set of N tasks based on the the selected periods, jitter values, and the execution time distribution. The utilization of the raw data, u, is then computed by Equation 24. Finally, the utilization value of the raw data is scaled up or down to U by multiplying $\frac{U}{u}$ to the execution time of each generated task. As a consequence, we obtain a set of tasks with the specified (U,N) value.

For each combination of (U,N) in which U = 5%, 10%, 15%, ... 100% and N = 10, 20, and 30, we apply the scheme to generate 5000 cases of input data and use the three algorithms to solve them. The schedulability degree of each (U,N) combination for an algorithm is obtained by dividing the number of solved cases by 5000. Since the jitter values is 1/10 of periods, it is observed that the SPF and SJF algorithms yield the same results. The results are shown in Figure 6.

As can be seen in Figures 6(a) and (b) the number of tasks has the different effects on the three algorithms. For SLsF, given a fixed utilization value, the schedulability degree increases

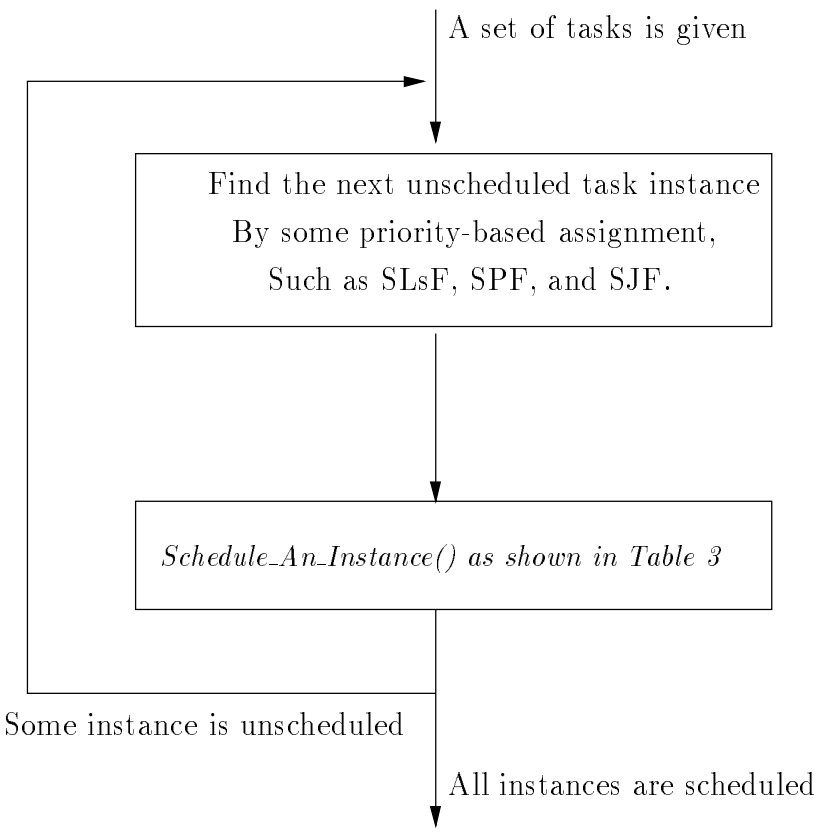

Figure 5: A schematic flowchart for the solution

that all the instances of τ_i are scheduled. The algorithm terminates when all counters overflow.

We can compute the relative deadline for a task instance by adding the execution time to the *lst*. If the execution times for all tasks are identical, the SLsF algorithm is equivalent to the *earliest deadline first* (EDF) algorithm.

5.2 SPF

The task periods determine the LCM of Γ and the numbers of instances for tasks within the LCM. In the most cases, the task with the smaller period has the tighter timing constraints. Namely, $(\lambda_i + \eta_i) \leq (\lambda_j + \eta_j)$ if $p_i \leq p_j$. To make the tasks with the smaller periods meet their timing constraints, the SPF algorithm favors the tasks with smaller periods.

The SPF algorithm uses the period as the key to arrange all tasks in non-decreasing order. The task with the smallest period is selected to schedule first. The instances of a particular task are scheduled one by one by invoking the algorithm in Table 3. After all the instances of a task are scheduled, the next task in the sequence is scheduled.

5.3 SJF

We define the *jitter* of a task τ_i as $(\lambda_i + \eta_i)$. It is proportional to the range of the scheduling window. Hence, The schedulability of a task also depends on the jitter.

Instead of using the period as the measurement, the SJF algorithm assigns the higher priority to the tasks with the smaller jitters. The task with the smallest jitter is scheduled first.

5.4 The Solution

The composition of the time-based scheduling of a task instance and the priority assignment of task instances is shown in Figure 5. The priority assignment can be done by using SLsF, SPF, or SJF. The function *Schedule_An_Instance()* is invoked to schedule a single task instance.

6 Experimental Evaluation

We conduct two experiments to study and compare the performance of the three algorithms. The purpose of the first experiment is to study the effect of the number of tasks and utilization on

smaller than $lst' - s_a^b$. In this case, Right_Shift(w_{k+1},$lst' - s_a^b - st(w_{k+1}) + f_a^b$) is invoked.

We do not enlarge an interval at both ends. Enlarging an interval at both ends needs to shift certain amount of preceding time slots to the left and shift some succeeding slots to the right. It is possible that some task instance τ_x^y is shifted left, while τ_x^{y+1} is shifted right. As a consequence, the timing constraints between s_x^y and s_x^{y+1} could be violated. For example, Let s_x^y and s_x^{y+1} before the shifting be 10 and 20 respectively. The execution time for τ_x is 5 time units. Assume the *left laxity* of τ_x^y is 5 and the *right laxity* of τ_x^{y+1} is 5. It implies $s_x^{y+1} - s_x^y \leq 15$. Consider the scheduling of a task instance τ_i^j with execution time 15. If we enlarge the interval between τ_x^y and τ_x^{y+1} by shifting τ_x^y left 5 time units and τ_x^{y+1} right 5 time units, then we get a new interval with 15 time units for τ_i^j. However, it turns out that $s_x^{y+1} = 25$, $s_x^y = 5$, and the relative timing constraints between τ_x^y and τ_x^{y+1} is violated.

5 The Priority-Based Scheduling of a Task Set

We consider the priority-based algorithms for scheduling a set of periodic tasks with hybrid timing constraints. Given a set of periodic tasks $\Gamma = \{ \tau_i \mid i = 1, \ldots, n \}$ with the task characteristics described in Section 2, we compute the LCM of all periods. Each task τ_i is extended to n_i task instances: τ_i^1, τ_i^2, ..., $\tau_i^{n_i}$. A scheduling algorithm σ for Γ is to totally order the instances of all tasks within the LCM. Namely, σ : *task_id* $\times$ *instance_id* $\rightarrow$ *integer*.

Three algorithms are considered. They are *smallest latest-start-time first* (SLsF), *smallest period first* (SPF), and *smallest jitter first* (SJF) algorithms.

5.1 SLsF

The scheduling window for a task instance τ_i^j depends on the scheduling of its preceding instance. Once s_i^{j-1} is determined, the scheduling window of the instance can be computed by equations 5 and 6. The scheduling window for the first instance of a task τ_i is defined as $[r_i,d_i - e_i]$.

The idea of the SLsF algorithm is to pick one candidate instance with the minimum *lst* among all tasks at a time. One counter for each task is maintained to indicate the candidate instance. All counters are initialized to 1. Each time when a task instance with the smallest *lst* is chosen, the algorithm in Table 3 is invoked to schedule the instance. After the scheduling of the instance is done, the counter is increased by one. The counter for τ_i overflows when it reaches $n_i + 1$. It means

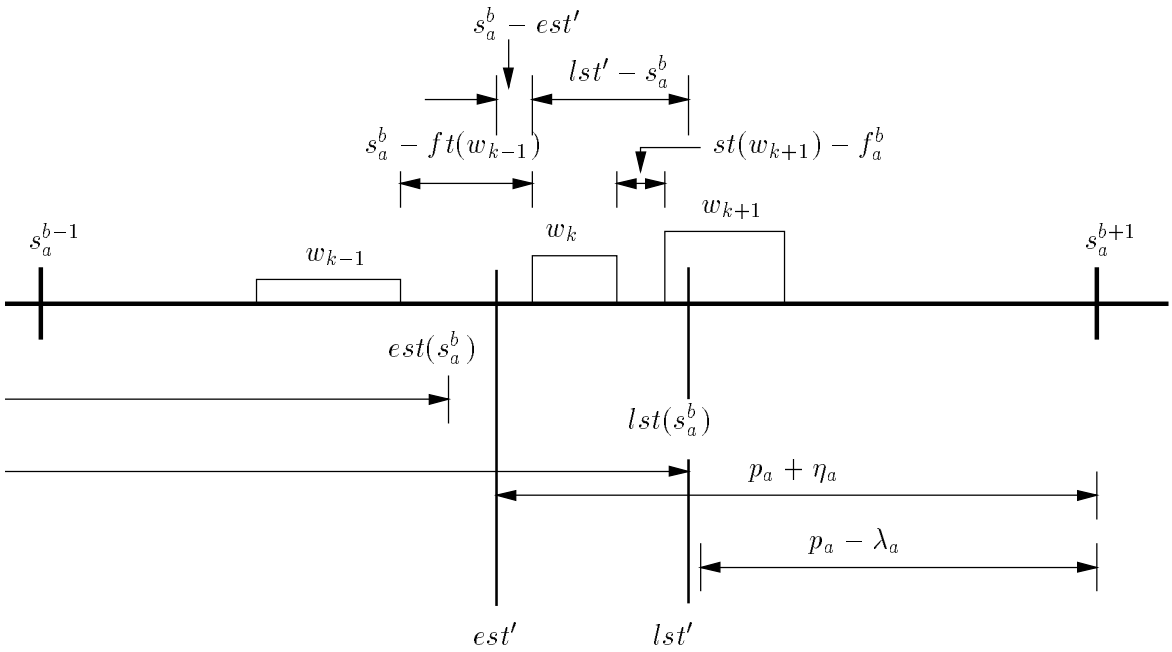

Figure 4: An illustration of $left_laxity(w_k)$ and $right_laxity(w_k)$

is needed to compute the laxity values for all time slots. The complexity is $O(2N)$ where N is the number of time slots in the linked list.

The basic idea of the sliding technique is described as follows. Given a task instance τ_i^j and a set of unoccupied intervals, $S = \{int_1, int_2, \ldots, int_{|S|}\}$, we check one interval at a time to see if the interval can be enlarged by shifting the existent time slots. Two possible ways of enlargement are (1) by either shifting the time slots, that precede the interval, to the left or (2) shifting the slots, that follow the interval, to the right. The shifting depends on which direction minimizes the objective function in Equation 21.

4.3 The Algorithm

An algorithmic description about how to schedule a task instance, as described in Sections 4.1 and 4.2, is given in Table 3.

The procedures Left_Shift(w_k,time_units) and Right_Shift(w_k,time_units) in Table 3 may involve the shifting of more than one time slot recursively. For example, consider the case in Figure 4, if Right_Shift(w_k,$lst' - s_a^b$) is invoked (i.e. w_k is to be shifted right by $lst' - s_a^b$ time units), then w_{k+1} has to be shifted too. It is because the gap between w_k and w_{k+1} is $st(w_{k+1}) - f_a^b$ which is

The idea is that if τ_i^j is scheduled to int_k, then the value in equation 21 will be smaller than that of the case in which τ_i^j is scheduled to int_{k+1}.

The scheduling of τ_i^j can be described as follows. Starting from int_1, we check whether the length of the interval is greater or equal to the execution time of τ_i^j or not. If yes, then we schedule the instance to the interval. One new time slot is created in which the start time is the lower bound of the interval and the finish time equals the start time plus the execution time. The created time slot is added to the linked list and the scheduling is done. If the length is smaller than the execution time, then we check the length of the next interval until all intervals are examined. An example is shown in Figure 3 in which the slot with dark area represents τ_i^j. In this example we assume that $est(\tau_i^j) \leq f_1$ and $s_2 - f_1 > e$. It means the free slot between the first and second occupied slots can be assigned to τ_i^j.

4.2 Sliding of the Time Slots

In case none of the intervals in S can accommodate a task instance, the sliding technique is used to create a big enough interval by sliding the existence time slots in the list.

To make the sliding technique work, we maintain two values for each time slot: *left laxity* and *right laxity*. The value of left laxity indicates the amount of time units by which a time slot can be left-shifted to a earlier start time. Similarly, the right laxity indicates the amount of time units by which a time slot can be right-shifted to a later start time.

Given the time slots w_{k-1}, w_k, and w_{k+1}, where a and b are the task and instance identifiers of w_k respectively, the laxity values of the time slot w_k can be computed by:

$$left_laxity(w_k) = min\{s_a^b - est',\ s_a^b - ft(w_{k-1}) + left_laxity(w_{k-1})\} \quad (22)$$

$$right_laxity(w_k) = min\{lst' - s_a^b,\ st(w_{k+1}) - f_a^b + right_laxity(w_{k+1})\} \quad (23)$$

$$\text{where} \quad est' = max\{est(\tau_a^b),\ s_a^{b+1} - (p_a + \eta_a)\}$$

$$\text{and} \quad lst' = min\{lst(\tau_a^b),\ s_a^{b+1} - (p_a - \lambda_a)\}.$$

Note that the interval $[est',\ lst']$ defines the sliding range during which τ_a^b can start without shifting τ_a^{b-1} or τ_a^{b+1}. A schematic illustration of equations 22 and 23 is given in Figure 4.

From equations 22 and 23, we see that the computing of *left_laxity*(w_k) depends on that of w_{k-1} and the computing of $right_laxity(w_k)$ depends on that of w_{k+1}. It implies a two-pass computation

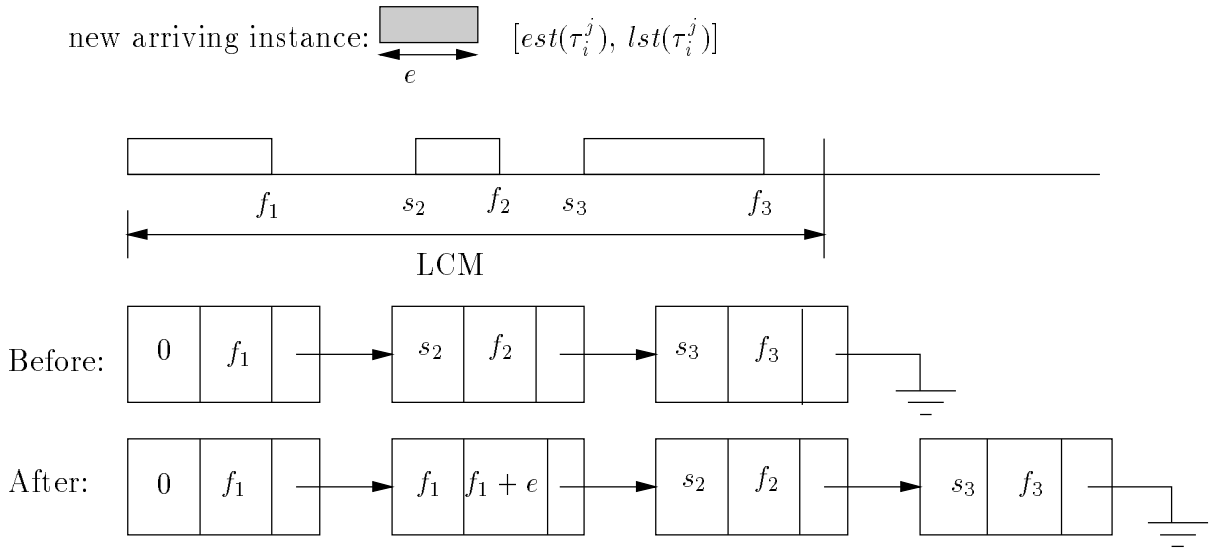

Figure 3: Insertion of a new time slot

pointers to the preceding and succeeding time slots respectively. We arrange the time slots in the list in increasing order by using the *start time* as the key. Any two time slots are non-overlapping. Since the execution of an instance is non-preemptable, the time difference between *start time* and *finish time* equals the execution time of the task.

4.1 Creating a Time Slot for the Task Instance

Consider a set of n tasks. Given a linked list and a task instance τ_i^j, we schedule the instance by inserting a time slot to the list. According to equations 5 and 6, we compute the $est(\tau_i^j)$ and $lst(\tau_i^j)$ first. Let S be the set of unoccupied time intervals that overlap the interval $[est(\tau_i^j),\ lst(\tau_i^j)]$ in the linked list. The unoccupied time intervals in S are collected by going through the list. Each time when a pair of time slots $(w, w+1)$ is examined, we compute $\ell = \max\{est(\tau_i^j),\ ft(w)\}$ and $\mu = \min\{lst(\tau_i^j),\ st(w+1)\}$, where $ft(w)$ is the finish time of the time slot w, and $st(w+1)$ is the start time of the slot next to w. If $\ell \leq \mu$, then we add the interval $[\ell,\ \mu]$ to S.

The free intervals in S are the potential time slots which τ_i^j can be assigned to. Since we try to schedule τ_i^j as close to one period away from the preceding instance τ_i^{j-1} as possible, we sort S, based on the function of the lower bound of each interval, $\alpha(s_i^{j-1} + p_i - \ell)$, in ascending order. Without loss of generality, we assume that S after the sorting is denoted by $\{int_1,\ int_2,\ \ldots,\ int_{|S|}\}$

$$
\begin{aligned}
&\leq \underline{s_i^1} + (j-1) \times p_i + (n_i - j + 1) \times \lambda_i \\
\text{and} \quad s_i^j &\geq \underline{s_i^{n_0}} + (j-1) \times p_i - (n_i - j + 1) \times \eta_i \underline{-(p_i + \eta_i) \times n_0 - 1}. \\
&\geq \underline{s_i^1} + (j-1) \times p_i - (n_i - j + 1) \times \eta_i
\end{aligned}
$$

Hence $P_i(j, y, n_i + n_0)$ holds for any $1 \leq n_0 \leq j$.

□

Theorem 1 *Given* s_i^1, s_i^2, ..., *and* s_i^j, *if,* $\forall\ k = 2, \ldots, j$, $\text{est}(\tau_i^k) \leq s_i^k \leq \text{lst}(\tau_i^k)$ *as specified in Equations 5 and 6, then* $P_i(j, y, z)$ *is true,* $\forall\ y = j+1, j+2, \ldots, n_i$, *and* $z = n_i + 1, n_i + 2, \ldots, n_i + j$.

By combining the proofs in Lemmas 1 and 2, it is easy to see that Theorem 1 holds. Based on Theorem 1 , we can assign the scheduling window for τ_i^j by using Equations 5 and 6 once s_i^1, s_i^2, ..., s_i^{j-1}.

Before we present the scheduling technique for a task instance, let us consider the following objective. The objective can be formulated as follows. Given a set of tasks with the characteristics described in Section 2, we schedule the task instances for each task within one LCM to minimize

$$
\begin{aligned}
\pi = \quad & \sum_{i,j} \alpha(s_i^j - s_i^{j-1} - p_i) \qquad (21) \\
\text{Subject to} \quad & \text{the constraints specified in Equations 1 through 4,} \\
& \text{where } \alpha(x) = x, \text{ if } x \geq 0; = -x, \text{ otherwise.}
\end{aligned}
$$

Basically, we try to schedule every instance of a task one period apart from its preceding instance. An optimal schedule is a feasible schedule with the minimum total deviation value from one period apart for instances.

4 The Time-Based Scheduling of a Task Instance

We consider the time-based solution to the scheduling problem by using a linked list. Each element in the list represents a time slot assigned to a task instance. A time slot w has the following fields: (1) *task id* i and *instance id* j indicate the identifier of the time slot. (2) *start time* st and *finish time* ft indicate the start time and completion time of τ_i^j respectively. (3) *prev ptr* and *next ptr* are the

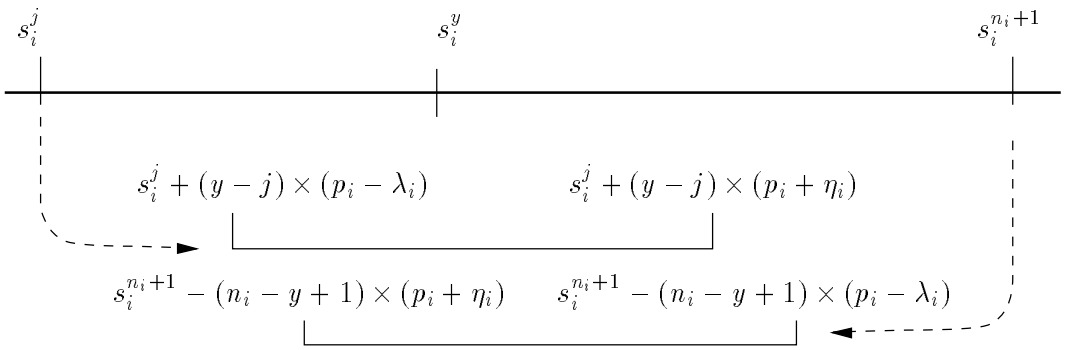

Figure 2: The overlapping of two intervals

□

Lemma 2 *Given s_i^1, s_i^2, ..., s_i^j, and an integer n_0, where $1 \leq n_0 \leq j$, if, $\forall\ k = 2, \ldots, j$, $est(\tau_i^k) \leq s_i^k \leq lst(\tau_i^k)$ are specified as in Equations 5 and 6, then $P_i(j, y, n_i + n_0)$ is true, $\forall\ y = j+1, j+2, \ldots, n_i$.*

Proof: We use the same method in Lemma 1 to prove it. We show that (1) the estimated scheduling window of s_i^y, based on s_i^j, is specified by the interval

$$[s_i^j + (y-j) \times (p_i - \lambda_i), s_i^j + (y-j) \times (p_i + \eta_i)], \tag{17}$$

(2) the estimated scheduling window of s_i^y, based on $s_i^{n_i+n_0}$, is specified by the interval

$$[s_i^{n_i+n_0} - (n_i + n_0 - y) \times (p_i + \eta_i), s_i^{n_i+n_0} - (n_i + n_0 - y) \times (p_i - \lambda_i)], \tag{18}$$

and (3) these two intervals overlap.

The following conditions have to be satisfied to make sure the overlapping of the two intervals.

$$s_i^j \leq s_i^{n_0} + (j-1) \times p_i + (n_i - j + 1) \times \lambda_i - (p_i - \lambda) \times n_0 - 1 \tag{19}$$

$$\text{and } s_i^j \geq s_i^{n_0} + (j-1) \times p_i - (n_i - j + 1) \times \eta_i - (p_i + \eta_i) \times n_0 - 1. \tag{20}$$

Since $s_i^1 \leq s_i^{n_0} - (p_i - \lambda) \times (n_0 - 1)$ and $s_i^1 \geq s_i^{n_0} - (p_i + \eta_i) \times (n_0 - 1)$, we rewrite Equations 19 and 20

$$s_i^j \leq \underline{s_i^{n_0}} + (j-1) \times p_i + (n_i - j + 1) \times \lambda_i \underline{- (p_i - \lambda) \times n_0 - 1}$$

Instance ID	*est* from Equation 5	*lst* from Equation 6	actual start time (s_i^j)
τ_i^1	0	40	4
τ_i^2	39	49	40
τ_i^3	75	85	77
τ_i^4	114	122	115
τ_i^5	159	160	$159 \sim 160$

Table 2: The correct setting of scheduling windows based on Proposition 3.1.

Lemma 1 *Given s_i^1, s_i^2, ..., and s_i^j, if, $\forall\ k = 2, \ldots, j$, $\text{est}(\tau_i^k) \leq s_i^k \leq \text{lst}(\tau_i^k)$ as specified in Equations 5 and 6, then $P_i(j, y, n_i + 1)$ is true, $\forall\ y = j+1, j+2, \ldots, n_i$.*

Proof: We prove that the estimated *est* and *lst* of τ_i^y, based on s_i^j and $s_i^{n_i+1}$, specify a feasible scheduling window, by showing that (1) the estimated scheduling window of s_i^y, based on s_i^j, is specified by the interval

$$[s_i^j + (y-j) \times (p_i - \lambda_i), s_i^j + (y-j) \times (p_i + \eta_i)], \tag{13}$$

(2) the estimated scheduling window of s_i^y, based on $s_i^{n_i+1}$, is specified by the interval

$$[s_i^{n_i+1} - (n_i - y + 1) \times (p_i + \eta_i), s_i^{n_i+1} - (n_i - y + 1) \times (p_i - \lambda_i)], \tag{14}$$

and (3) the intervals in Equations 13 and 14 overlap.

In Figure 2, we see that the necessary and sufficient conditions for the overlapping of the intervals specified in Equations 13 and 14 are

$$s_i^j + (y-j) \times (p_i - \lambda_i) \leq s_i^{n_i+1} - (n_i - y + 1) \times (p_i - \lambda_i) \tag{15}$$

$$\text{and } s_i^{n_i+1} - (n_i - y + 1) \times (p_i + \eta_i) \leq s_i^j + (y-j) \times (p_i + \eta_i). \tag{16}$$

By solving the Equations 15 and 16, we obtain

$$\begin{aligned} s_i^j &\leq s_i^1 + (j-1) \times p_i + (n_i - j + 1) \times \lambda_i \\ \text{and } s_i^j &\geq s_i^1 + (j-1) \times p_i - (n_i - j + 1) \times \eta_i. \end{aligned}$$

The above two equations describe the same conditions as Equations 11 and 12 do. Hence, $P_i(j, y, n_i + 1)$ is true, $\forall\ y = j+1, j+2, \ldots, n_i$.

$$s_i^{n_i+1} - \mu \leq p_i + \eta_i \tag{10}$$

By replacing ℓ in Equation 9 with $s_i^j + (n_i - j) \times (p_i - \lambda_i)$, we obtain

$$\begin{aligned} s_i^j &\leq s_i^{n_i+1} - (n_i - j + 1) \times (p_i - \lambda_i) \\ &= s_i^1 + \text{LCM} - (n_i - j + 1) \times (p_i - \lambda_i) \\ &= s_i^1 + n_j \times p_i - (n_i - j + 1) \times (p_i - \lambda_i) \\ &= s_i^1 + (j-1) \times p_i + (n_i - j + 1) \times \lambda_i \end{aligned} \tag{11}$$

Likewise, by replacing μ in Equation 10 with $s_i^j + (n_i - j) \times (p_i + \eta_i)$, we have

$$\begin{aligned} s_i^j &\geq s_i^{n_i+1} - (n_i - j + 1) \times (p_i + \eta_i) \\ &= s_i^1 + \text{LCM} - (n_i - j + 1) \times (p_i + \eta_i) \\ &= s_i^1 + (j-1) \times p_i - (n_i - j + 1) \times \eta_i \end{aligned} \tag{12}$$

So, According to Equations 12 and 3, we choose the bigger value between $(s_i^{j-1} + p_i - \lambda_i)$ and $(s_i^1 + (j-1) \times p_i - (n_i - j + 1) \times \eta_i)$ as the *est* of τ_i^j. Similarly, according to Equations 11 and 4, we assign the smaller value of $(s_i^{j-1} + p_i + \eta_i)$ and $(s_i^1 + (j-1) \times p_i + (n_i - j + 1) \times \lambda_i)$ as the *lst*.

□

Example 3.1: To show how Proposition 3 gives a tighter bound to find feasible scheduling windows, we consider the case shown in Table 1 again. We apply Equations 5 and 6 to compute the *est* and *lst* of each instance. The results are shown in Table 2. Note that the scheduling windows for τ_i^4 and τ_i^5 are tighter than those in Table 1. As a consequence, any start time in the interval [159,160] for τ_i^5 satisfys the relative timing constraints between τ_i^5 and τ_i^6.

3.1 Property of Scheduling Windows

Define $P_i(x, y, z)$ as the predicate in which the estimated *est* and *lst* of τ_i^y, based on s_i^x and s_i^z, specify a feasible scheduling window for τ_i^y. In Proposition 3 , we prove that for any s_i^j in between $est(\tau_i^j)$ and $lst(\tau_i^j)$ as specified in Equations 5 and 6, $P_i(j, n_i, n_i + 1)$ is true.

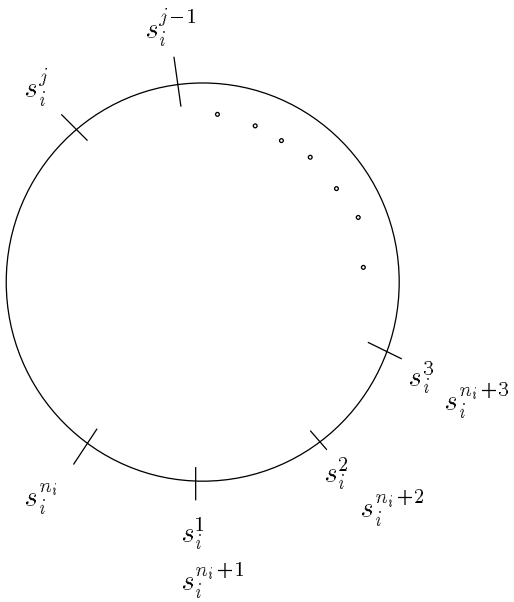

Figure 1: The relations between the task instances

$[s_i^{n_i+1} - (p_i + \eta_i),\ s_i^{n_i+1} - (p_i - \lambda_i)]$.

Proposition 1: Let the est and lst of τ_i^j be

$$est(\tau_i^j) = max\{(s_i^{j-1} + p_i - \lambda_i),\ (s_i^1 + (j-1) \times p_i - (n_i - j + 1) \times \eta_i)\}, \quad (5)$$

$$\text{and } lst(\tau_i^j) = min\{(s_i^{j-1} + p_i + \eta_i),\ (s_i^1 + (j-1) \times p_i + (n_i - j + 1) \times \lambda_i)\}. \quad (6)$$

If s_i^j is in between the $est(\tau_i^j)$ and $lst(\tau_i^j)$, then the estimated *est* and *lst* of $s_i^{n_i}$, based on s_i^j and $s_i^{n_i+1}$, specify a feasible window.

Proof: Let ℓ and μ be the estimated *est* and *lst* of $s_i^{n_i}$, based on s_i^j, respectively.

Hence,

$$\ell = s_i^j + (n_i - j) \times (p_i - \lambda_i) \quad (7)$$

$$\mu = s_i^j + (n_i - j) \times (p_i + \eta_i) \quad (8)$$

To guarantee the existence of feasible start time of $\tau_i^{n_i}$, the interval $[\ell,\mu]$ has to overlap the interval $[s_i^{n_i+1} - (p_i + \eta_i),\ s_i^{n_i+1} - (p_i - \lambda_i)]$. Hence the following conditions have to be satisfied:

$$s_i^{n_i+1} - \ell \geq p_i - \lambda_i \quad (9)$$

Instance ID	$est = s_i^{j-1} + p_i - \lambda_i$	$lst = s_i^{j-1} + p_i + \eta_i$	actual start time (s_i^j)
τ_i^1	0	40	4
τ_i^2	39	49	40
τ_i^3	75	85	77
τ_i^4	112	122	113
τ_i^5	148	158	*

Table 1: An example to show the wrong setting of scheduling windows

We consider the scheduling of periodic tasks with relative timing constraints described in Equations 3 and 4. The scheduling window for a task instance is derived from the start times of its preceding instances. A *feasible* scheduling window for a task instance τ_i^j is a scheduling window in which any start time in the window makes the timing relation between s_i^{j-1} and s_i^j satisfy Equations 3 and 4. Formally, given s_i^1, s_i^2, ..., and ..., s_i^{j-1}, the problem is to derive the feasible scheduling window for τ_i^j such that a feasible schedule can be obtained if τ_i^j is scheduled within the window.

For the sake of simplicity, we assume that $r_i = 0$ and $d_i = p_i$, $\forall\ i$, in this section. Then, simply assigning *est* and *lst* of τ_i^j as $s_i^{j-1} + p_i - \lambda_i$ and $s_i^{j-1} + p_i + \eta_i$ respectively where $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, n_i$, is not tight enough to guarantee a feasible solution. For example, consider the case shown in Table 1 in which a periodic task τ_i is to be scheduled. Let LCM, p_i, λ_i, and η_i be 200, 40, 5, and 5 respectively. Hence, there are 5 instances within one LCM (i.e. $n_i = 5$). The first column in Table 1 indicates the instance IDs. The second and third columns give the *est* and *lst* of the scheduling windows for the task instances specified in the first column. The last column shows the actual start times scheduled for the particular task instances. The actual start time is a value in between *est* and *lst* of each task instance. For instance, the *est* and *lst* of τ_i^2 are 39 and 49 respectively. It means $39 \leq s_i^2 \leq 49$. The scheduled value for s_i^2, in the example, is 40. Since $s_i^6 = s_i^1 + \text{LCM} = 204$, we find that any value in the interval [148,158] can not satisfy the relative timing constraints between τ_i^5 and τ_i^6. As a consequence, the constructed schedule is infeasible.

We draw a picture to depict the relations among the start times of task instances in Figure 1. When τ_i^j is taken into account, the scheduling window for s_i^j is obtained by considering its relation with s_i^{j-1} as well as that with $s_i^{n_i}$ and $s_i^{n_i+1}$. We make sure that once s_i^j is determined, the estimated *est* and *lst* of $s_i^{n_i}$, based on s_i^j and $s_i^{n_i+1}$, specify a feasible scheduling window for $s_i^{n_i}$. Namely, the interval which is specified by the estimated *est* and *lst* of $s_i^{n_i}$, based on s_i^j, overlaps the interval

2 Problem Statement

Consider a set of periodic tasks $\Gamma = \{ \tau_i \mid i = 1, \ldots n \}$, where τ_i is a 4-tuple $< p_i, e_i, \lambda_i, \eta_i >$ denoting the period, computation time, low jitter and high jitter respectively. One instance of a task is executed each period. The execution of a task instance is non-preemptable. The start times of two consecutive instances of task τ_i are at least $p_i - \lambda_i$ and at most $p_i + \eta_i$ apart.

In order to schedule periodic tasks, we consider the least common multiple (LCM) of all periods of tasks. Let n_i be the number of instances for task τ_i within a schedule of length LCM. Hence, $n_i = \frac{\text{LCM}}{\text{p}_\text{i}}$. A schedule for a set of tasks is the mapping of each task τ_i to n_i task instances and the assigning of a start time s_i^j to the j-th instance of task τ_i, τ_i^j, $\forall\ i = 1, \ldots n$ and $j = 1, \ldots, n_i$. A *feasible* schedule is a schedule in which the following conditions are satisfied for each task τ_i:

$$
\begin{aligned}
f_i^j &= s_i^j + e_i && (1)\\
s_i^{n_i+1} &= s_i^1 + \text{LCM} && (2)\\
s_i^j &\geq s_i^{j-1} + p_i - \lambda_i && (3)\\
s_i^j &\leq s_i^{j-1} + p_i + \eta_i && (4)\\
&\quad \forall j = 2, \ldots, n_i + 1.
\end{aligned}
$$

The non-preemption scheduling discipline leads to Equation 1 where f_i^j is the finish time of τ_i^j. Another condition for non-preemption scheduling is that given any i, j, k and ℓ, if $s_i^j < s_k^\ell$ then $f_i^j \leq s_k^\ell$. It means the schedule for any two instances is non-overlapping. The constructed schedule of length LCM is invoked repeatedly by wrapping-around the end point of the first schedule to the start point of the next one. Hence, as shown in Equation 2, the start time of the first instance in the next schedule is exactly one LCM away from that of the first schedule. Finally, Equations 3 and 4 specify the relative timing constraints between two consecutive instances of a task.

3 Analysis of Relative Timing Constraints

Define the scheduling window for a task instance as the time interval during which the task can start. Traditionally, the lower and upper bounds of the scheduling window for a task instance are called earliest start time (*est*) and latest start time (*lst*) respectively. These values are given and independent of the start times of the preceding instances.

1 Introduction

The task scheduling problem is one of the basic issues of building real-time applications in which the tasks of applications are associated with timing constraints. For the hard real-time applications, such as avionics systems and nuclear power systems, the approach to guarantee the critical timing constraints is to schedule periodic tasks *a priori*. A non-preemptive schedule for a set of periodic tasks is generated by assigning a start time to each execution of a task to meet their timing constraints. Failure to meet the specified timing constraints can result in disastrous consequence.

Various kinds of periodic task models have been proposed to represent the real-time system characteristics. One of them is to model an application as a set of tasks, in which each task is executed once every period under the ready time and deadline constraints. These constraints impose constant intervals in which a task can be executed. In literature, many techniques [2, 3, 4, 5, 6, 7, 8] have been proposed to solve the scheduling problem in this context. The deficiency of this modeling is the inability of specifying the relative constraints across task periods. For example, one can not specify the timing relationship between two consecutive executions of the same task.

Simply assuring that one instance of each task starts the execution after the ready time and completes the execution before the specified deadline is not enough. Some real-time applications have more complicated timing constraints for the tasks. For example, the relative timing constraints may be imposed upon the consecutive executions of a task in which the scheduling of two consecutive executions of a periodic task must be separated by a minimum execution interval. The Boeing 777 Aircraft Information Management System is such an example [1]. One possible solution to the scheduling problem of such applications is to consider the instances of tasks rather than the tasks. A task instance is defined as one execution of a task within a period. With the notion of task instances, one is able to specify the various timing constraints and dependencies among instances of tasks.

In this paper, we consider the relative timing constraints imposed on two consecutive instances of a task. The task model and the analysis of the timing constraints are introduced in Sections 2 and 3 respectively. Based on the analysis, we are able to derive the scheduling window for each task instance. Given the scheduling window of a task instance, we present the time-based approach of scheduling a task instance in Section 4. We propose three priority assignment algorithms for the task instances in Section 5. The task instances are scheduled one by one based on their priorities. In Section 6, we evaluate the three algorithms and show the experimental results.

Scheduling of Periodic Tasks with Relative Timing Constraints *

Sheng-Tzong Cheng and Ashok K. Agrawala
Institute for Advanced Computer Studies
Systems Design and Analysis Group
Department of Computer Science
University of Maryland
College Park, MD 20742
{stcheng,agrawala}@cs.umd.edu

Abstract

The problem of non-preemptive scheduling of a set of periodic tasks on a single processor has been traditionally considering the ready time and deadline on each task. As a consequence, a feasible schedule finds that in each period one instance of each task starts the execution after the ready time and completes the execution before the deadline .

Recently, the timing requirements of the real-time systems emerge that the relative timing constraints are imposed on the consecutive executions of each task. In this paper, we consider the scheduling problem of the periodic tasks with the relative timing constraints imposed on two consecutive executions of a task. We analyze the timing constraints and derive the scheduling window for each task instance. Based on the scheduling window, we present the time-based approach of scheduling a task instance. The task instances are scheduled one by one based on their priorities assigned by the proposed algorithms in this paper. We conduct the experiments to compare the schedulability of the algorithms.

*This work is supported in part by Honeywell under N00014-91-C-0195 and Army/Phillips under DASG-60-92-C-0055. The views, opinions, and/or findings contained in this report are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of Honeywell or Army/Phillips.