Project Report of CSCE790i

Performance of Algorithms for Scheduling Soft Real-Time System

Linwei Niu
Department of Computer Science & Engineering
University of South Carolina
Columbia, SC 29208
niul@cse.sc.edu

Abstract

In this research project the performance of two well-known scheduling policies for Real Time Systems based on fixed priority, namely Enhanced (m,k)-firm Guarantee and Weakly hard scheduling, will be compared in terms of the overall schedulibility of a number of randomly generated task sets . Both methods are based on the common assumptions. A new scheduling alogrithm will then be derived by combining the advantages of both algorithm for the purpose of improving the overall schedulibility of the real time system.

Keywords Scheduling, task, job, Real-time system, Weakly hard, Constraints, Schedulability, Interference, WCIP, Independent Job .

1 Introduction

In some multimedia real time system like video conference system, the loss of some image frames due to their missing of deadlines are tolerable. This class of system are sometimes refer to as soft real-time system or firm real time system.

A real time system is said to have (m,k)-firm deadlines if at least m out of any k consecutive jobs from the task(or, say, a job chain) must meet their deadlines. Similarly, the concept of weakly hard real-time system refers to the real-time system that can tolerate a clearly specified number of deadlines

In[9], Ramanathan first proposed a scheduling technique that can provide the deterministic (m,k)-firm guaranntee. In this techniqhe, a simple algorithm is adopted to divide the jobs of each task into two categories: mandatory and optional. All mandatory jobs are required to meet their deadline. The advantages of this method is that it gives a simple formula to partition the jobs into mandatory or optional and the (m,k)-firm guarantee requirement is satisfied if all mandatory instances meet their deadlines. In[8], Quan developed the idea in[1] by adjusting the relative position of the mandatory jobs of each task. The beauty of this technique is that it reflects part of the real execution situation between jobs in terms of execution interference and by shifting the WCIPs of each task away, the execution interference from higher priority tasks with lower priority tasks was reduced. Thus the worst case response time of mandatory jobs is reduced. Thereafter the schedulibility of the whole task set is improved. However, the problem of evenly distribution of mandatory jobs still exists and the relative positions of mandatory jobs of different values of m and k with the same ratio keeps the same. So some tasks might be designated as "optional" even when they would meet their deadlines if they were scheduled and conversely, some tasks might be designated as "mandatory" and invoked even when they would miss their deadlines.In[2], Bernat uses a method based on precise timing analysis to decide the schedulibility of a given task set.

The difference between these two method is that for the former one, only the mandatory job are invoked and any missing deadline will cause dynamic failure of the whole task set. The analysis of the schedulibility of task sets is based on the analysis of critical instance of each task. Similar work has been done by Lehoczky in analyzing hard schedulibility of task sets with fixed priorities. The basic idea is that if the critical instance of each task can meet the deadline, then the whole task set is schedulable. For the latter one, all jobs of each task are invoked and runs until completion even though it misses the deadline. The schedulibility of the task set is tested by offline computation on the response time of each job of each task throughout its hiperperiod and the constraint is checked based on the computation result. The advantage of this method is that precise timing ananlysis is done on the jobs of each of the task in respect to computing the values of Fi(k), Si(k),Ri(k). Thus, the exact running status of each job can be predicted and more information on the execution situation can be achieved by

an offline computation. Moreover, the precise quantification method in this algorithm can be used widely in the real-time system area with slightly modification. Although the computation of it may consume considerable time, it is not a major concern for offline computation. The disadvantage is that since it adopt delayed finalization to handle the missing of deadline. All jobs will be supposed to continue running until it finishes even though it may miss the deadline. This has two consequences: the first one is that part the processor time was occupied by those jobs missing the deadline(if they do) and this not only degrade the system efficiency but also imposed some limit on the utilization bound that algorithm can process. For the prevailing cases in overloaded system where the resource utilization is larger than 1, the algorithm will face the embarrassment that it can not guarantee the test has covered all the worst case response time of jobs and thus cannot tell whether the weakly hard constraint can be satisfied.; the second one is the so called "domino effect" which is also a typical phenomenon that may happen with EDF when the system is overloaded. Similar with EDF, this happens because the delayed finalization of the first job make the following jobs even harder to schedule and cause them to miss the deadline. However, if the two algorithms are combine together, better result can be expected in improving the overall schedulability of the system. That's a main work of this project.

This rest part of this report is organized as followed: in section2, we will introduce the process model of our system; in section 3, we introduce the algorithms to be compared in specification; in section 4, we gave the experiment results and our analysis on it; in section 5, we consider real time systems with variable execution time and arbitrary deadlines; in section 6, we draw the conclusion based on what we've done in the previous sections.

2 System and Process Model

Before we describe the sheduling algorithms we have considered in our perfomance study, we define the following notations to refer to the parameters of a task set with n independent periodic tasks, $\tau = \{\tau_1, \tau_2, \cdots, \tau_n\}$, arranged in the decreasing order of their priorities.Each instance of task is called a *job*. The jth job of τ_i is denoted as τ_{ij}. The following notations are defined to refer to the parameters of task τ_i :

- O_i : the release time of the first job of τ_i, refered to as initial time.
- T_i : the interval between two consecutive job release times of τ_i, refered to as *period*.

- D_i : the maximum time allowed from the release to the completion of τ_i's job, refered to as deadline.
- C_i : the maximum time needed to complete τ_i without any interuption, refered to as execution time.
- m_i and k_i : the (m,k) constraint for τ_i, which mandates that at least m out of k consecutive jobs of τ_i must meet their deadlines to avoid any dynamic failure.

Other assumptions associated with the notations are:

- Deadlines for hard tasks cannot be missed;
- Task periods are fixed and cannot be modified;
- Deadline equals period (D = T);
- Optional(or soft) tasks are assigned the lowest priority

All tasks are assumed to be scheduled on a uniprocessor system and both of the methods adopt a fixed prioriy scheduler that always executes the runnable task with the highest priority . Static scheduling assumes that the complete knowledge regarding the task set and its constraints a priori and once a task's priority is a assigned it does not have to be reevaluated as time progresses. This has several advantages: (*i*) scheduling can often be done with an offline analysis. (*ii*)it incurs lower overhead; (*iii*) the implementation is relatively simple; (*iiii*) it gives a designer control over task priorities.

3 Specification of Algorithms to Compare

3.1 Algorithm 1 (Alg-WH) : Weakly Hard Scheduling Algorithm [2]

Definition 1. *The task's $\mu - pattern$ is the worst case pattern of met and lost deadlines of task τ_i. It is denoted by μ_i and it is defined as:*

$$\mu_i(k) = \begin{cases} 1 \; if \; R_i(k) \; \leq D_i \\ 0 \; Otherwise \end{cases} \quad (1)$$

where $R_i(k)$ is the worst case response time of the task at invocation $k \geq 1$, this valuse is given by $R_i(k) = F_i(k) - S_i(k)$, where $F_i(k)$ is the worst case finilization time of the k-th invocation of the task and $S_i(k)$ is the release time.

It has been show in [2] that if the utilization factor $U \leq 1$ then the worst case task $\mu - pattren$ is repeated every hyperperiod at level i,given by: $h_i = lcm\{T_j \mid 1 \leq j \leq i\}$.

Definition 2. *Bernat [2] A task τ is said to satisfy the weekly hard temporal constraint "meets any n in m deadlines" ($m \geq 1$ and $0 \leq n \leq m$), denoted by $\tau \vdash \begin{pmatrix} n \\ m \end{pmatrix}$ if, in any window of m consecutive invocations of the task, there are at least n invocations in any order that meet the deadline.(i.e., in any window of size m in the corresponding $u - pattern$, there are at least n 1s).*

This definition corresponds to the definition of (m,k)-firm model in the Alg-EG algorithm to be compared.

In order to check whether the tasks of a system satisfy their weakly hard specifications, we must compute the μ-pattern of each task τ_i. In[2], Bernat proved that if the utilization of the task set is not larger than 1, the μ-patterns are closed, so their analysis can be reduced to a finite sequence.Thus, we can check whether the task set meet the weakly hard constraint by analizing the μ-pattern with a sliding window of lengh k_i in its *Hyperperiod* [2].
newline

Theorem 1. *(Bernat [2]): Give a task set $T = \{\tau_1, \tau_2, \cdots, \tau_n\}$, made up of independent periodic tasks scheduled under fixed priority , then all tasks(and the $\mu - pattern$ correspondingly) are closed if and only if $U \leq 1$.*

Now, the problem is how to compute the $\mu - pattern$. In[2], Bernat gave the specification of doing that. Formally, we need to compute $F_i(k), R_i(k)$ of each job k within $2 * h_i + \max\{O_j\}$[1], where $F_i(k)$ is the minimum t that makes equation(2) hold; t is the time required to perform k computations of the task, plus the interference from higher priority tasks suffered during that period, plus the time the processor has not been used(or, say, the idle time at level i) from t=0 until the start time of the task by tasks with priority higher than or equal to τ_i. Formally,

$$t = kC_i + Intf_i(t) + Idle_i(S_i(k)) \tag{2}$$

The Interference function $Intf_i(t)$ is given by:

$$Intf_i(t) = \sum_{\tau_j \in hp(i)} \lceil \frac{t - O_j}{T_j} \rceil \ C_j \tag{3}$$

where $hp(i)$ is the set of task of higher priority than task τ_i.$Idle_i(t)$ is the time the processor can be used by lower priority tasks between [0, t). This can be computed by adding a virtual task $\bar{\tau} = (\bar{T} = t, \bar{D} = t, \bar{C}, \bar{O} = 0)$ of lower priority than τ_i and computing the maximum value $\bar{C}$, such that task $\bar{tau}$ meets its deadline. Formally,

$$Idle_i(t) = max\{ \ \bar{C} \mid \bar{\tau} \ is \ schedulable \ \} \tag{4}$$

The finish time for task is given by:

$$t = \bar{C} + \sum_{\tau_j \in hep(\tau_i)} \lceil \frac{t - O_j}{T_j} \rceil \ C_j \tag{5}$$

And checking if $t < \bar{D}$. $hep(\tau_i)$ is the set of tasks of higher or equal priority than τ_i.After we compute the $R_i(k)$ of each job k such that $S_i(k) \leq 2 * h_i + max\{O_j\}$, we can compute the corresponding $\mu - pattern$ according to (1) and check the result $\mu - patterns$ to test the schedulability of the task set.

In our algorithm3, we developed this method to make it fit into the computation of (m,k)-firm guarantee and use the new method in deciding the local optimal pattern of task τ_i.

3.2 Algorithm 2 (Alg-EG) : Enhanced (m,k)-firm Guarantee algorithm [8]

When scheuling a task set with (m,k) constraints according to a fixed-priority assignment, one critital step is to determine for each task wether its exectution is mandatory or optional.

Definition 3. *The (m,k)-pattern of task* τ_i *, denoted by* Π_i*, is a binary string* $\Pi_i = \{\pi_{i1}\pi_{i2} \cdots \pi_{ik_i} \}$ *which satisfies the following :* (*i*) π_{ij} *is a mandatory job if* $\pi_{ij} = 1$ *and optional if* $\pi_{ij} = 0$, *and* (*ii*) $\sum_{j=1}^{ki} \pi_{ij} = m_i$. By repeating the (m,k)-Pattern Π_i, we get a mandatory job pattern for τ_i. It's not difficult to see that the (m,k) constraints for τ_i can be satisfied if the mandatory jobs of τ_i are selected accordingly.

After given the (m,k) constraint, how to how to find the proper (m,k)-pattern to schedule the task set is a difficult problem. In [8], Quan has shown that selecting the "optimal" (m,k)-pattern for the given (m,k) constraint is NP-hard in the strong sense. In [9], Ramanathan proposed a simple algorithm to seletct the mandatory jobs and proved that so long

as the (m,k)-pattern is determined by the following formular, the (m,k)constraint is guaranteed.

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

In [8], Quan developed this idea by shifting away the WCIPs between tasks away and proposed a new algorithm which greatly improved the schedulibility of the task set. The way that Quan determines the (m,k)-pattern of the task set is to modify the formular(2) like this:

$$\pi_{ij} = \begin{cases} 1 & \text{if } j = \lfloor \lceil \frac{((j-1)+s_i)\times m_i}{k_i} \rceil \times \frac{k_i}{m_i} \rfloor + 1 \\ 0 & \text{otherwise} \end{cases} \qquad j = 1, 2, \cdots, k_i \tag{7}$$

where $s_i \leq 0$ and $s_i \in Z$. Note that the new (m,k)-pattern can be viewed as rotating the (m,k)-pattern in (2) right by s_i bits. It's easy to see the new (m,k)-pattern satisfies the (m,k)constraints as well.Quan also present the algorithm to compute the proper value of S_i in [8].

When it comes to checking the schedulability of a task set for a given set of (m,k)- patterns, Quan also presented a sufficient condition [8] to predict the schedulability of mandatory jobs in polynomial time. It's shown by the theorm below.

Theorem 2. *(Quan[8]): Given two task set T and T' with* $T_i^{'} \leq T_i$, $C_i^{'} = C_i, m_i^{'} = m_i, k_i^{'} = k_i$, *and* $T_j^{'}$ *divides* $T_i^{'}$ *if* $T_j^{'} \leq T_i^{'}$. *With the given* $(m,k) - patterns, if$ $\sum_{j \leq i}(l_{ij} \times C_j)/T_i^{'} \leq 1$, l_{ij} *is the maximum number of mandatory jobs during any time interval of length* $T_i^{'}$, *then T is schedulable.*

The advantage of this sufficient condition is that it takes only$O(n^k)$ time to implement so that it can tell a task set is schedulable quickly if it is. However, it also has the disadvantage that for some task sets with (m,k)-pattern given by (3), it can not tell they are schedulable while actually they are. This behaviour is somewhat similar to that of the Liu and layland's upper bound (the sufficient condition in[7]). Due to this obvious reason in [8] Quan used simulation to achieve the goal. The advantage of using simulation is that it's straightforward and easy to implement. The disadvantage is that it may take very long time. This is due to the fact that simulation has to run all tasks until the Hyperperiod[2](or twice the Hyperperiod if offsets are present);for non-trivial systems this can be very large. In [2],Bernat introduce a method based on offline computation to check the schedulability of weakly hard system and showed that it's more efficient. However, this method cannot be directly used on (m,k)-firm guarantee. Here

we developed the method in [2] to make it fit into the (m,k)-firm guaranntee and use it to check the schedulability of the task set offline.

We now show how to check whether each mandatory job can meet its' deadline. Before doing that, we need to show how to compute $F_i(k), R_i(k)$ of each job with the given (m,k)-pattern. To achieve this goal, we shall develop the method in Algorithm 1 by incorporating the (m,k)-pattern to it. Formally,given the (m,k)-pattern of a task set, we need to compute $F_i(k), S_i(k), R_i(k)$ of the mandatory jobs whithin the first hyperperiod at level i, where $F_i(k)$ is the minimum t that makes equation(8) hold; t is the time required to perform k computations of the task, plus the interference from higher priority tasks suffered during that period, plus the time the processor has not been used(or, say, the idle time at level i) from t=0 until the start time of the task by tasks with priority higher than or equal to τ_i. Formally,

$$t = \sum_{1 \leq\ l \leq\ k} \pi_{il} C_i + Intf_i(t) + Idle_i(S_i(k)) \tag{8}$$

The Interference function $Intf_i(t)$ is given by:

$$Intf_i(t) = \sum_{\tau_j \in hp(i)} \sum_{1 \leq l \leq\ \lceil \frac{t-O_j}{T_j} \rceil} \pi_{jl} C_j \tag{9}$$

where $hp(i)$ is the set of task of higher priority than task τ_i. $Idle_i(t)$ is the time the processor can be used by lower priority tasks between [0, t). This can be computed by adding a virtual task $\bar{\tau} = (\bar{T} = t, \bar{D} = t, \bar{C}, \bar{O} = 0)$ of lower priority than τ_i and computing the maximum value $\bar{C}$, such that task $\bar{\tau}$ meets its deadline. Formally,

$$Idle_i(t) = max\{\ \bar{C} \mid \bar{\tau}\ is\ schedulable\ \} \tag{10}$$

The finish time for task is given by:

$$t = \bar{C} + \sum_{\tau_j \in hep(i)} \sum_{1 \leq l \leq\ \lceil \frac{t-O_j}{T_j} \rceil} \pi_{jl} C_j \tag{11}$$

And checking if $t < \bar{D}$. $hep(\tau_i)$ is the set of tasks of higher or equal priority than τ_i.
Observe that τ_i's mandatory jobs corresponding to bit $\pi_{ij} = 1$ can be viewed as a periodic event E_i with period $k_i T_i$, the concept of hyperperiod at level i in algorithm 1 should also be modified correspondingly. It's given by:$h_i = lcm\{k_j T_j \mid 1 \leq j \leq i\}$.

In [1],Bernat showed that it's only necessary to compute $R_i(k)$ for values of k such that $S_i(k) \leq 2 * h_i + max\{O_j\}$.

3.3 Algorithm 3 (Alg-IG) : Improved (m,k)-firm Guarantee algorithm

This algorithm is a development of Alg-EG algorithm. As we mentioned before, although in most cases Alg-EG algorithm works better than Alg-WH algorithm(this will be shown by the experiment result), it has some limitation in partitioning the jobs, for example designating the some job that will miss the deadline as *mandantory* . Sometime maybe just "one job" is wrongly selected as *mandatory* and it causes the whole taske set unschedulable becaus it breaks the guarantee that *each* of the *mandatory* job meet the deadline. For example, suppose task τ_i has the following μ-pattern for each *independentjob* in its hyperperiod (the concept of independent job is introduced in section 3.4) :

"11001110111101111001"

Suppose it has the (m,k)-constraint as (3,5). Then the evenly ditributed pattern(e.g. "10101") does not work for this job while evenly ditributed pattern like "11001" works.
In this Algorithm, we will develop the timing ananlysis method in Alg-WH and then applied it to Alg-EG to achieve a better pattern. The origin of this idea is that in algorithm Alg-EG, the problem of implicit even distribution of pattern still exits because it still depends on the formular introduced in [9] to determine the pattern, which makes the pattern still not flexible. Although the shedulibility of the whole system is improved by shifting the WCIPs[8] away, there are still a lot of task sets that's are schedulable (e.g. the task the above example) but cannot be scheduled by Alg-EG. The reason is because only part of execution situation of the whole system is relected in Alg-EG . In oreder to improve the schedulibility of the whole task sets, specificlly, to schedule those schedulable task sets that cannot by scheduled by Alg-EG, more of the real execution situation need to be reflected in choosing the pattern. Fortunately, the precise timing ananlysis method in Alg-WH provides us a powerful tool in doing that. In [8], Quan proved that the problem of "selecting" the "optimal" (m,k) pattern for each task is $NP - hard$.And we conjecture the problem of finding the "optimal" *loop style (m,k)*-pattern for each task is also $NP - hard$. However,with our new method,the locally optimal *loop style (m,k)-pattern* of task τ_i can be decided easily. Before doing that, we need prove the theorem below first:

Theorem 3. *(loop pattern theorem) Given an (m,k)-firm constraint for a task τ_i, any loop*

pattern with m 1's out of k positions(k is the length of the piece of pattern to be looped) can meet the (m,k)-firm constraint.

proof: the proof of this theorem can be done with mathematical induction or simply by exhaustive enumeration for any given values of m and k. For example,give (m_i, k_i)-firm constraint as (3,5) for task τ_i, then any loop pattern based on loop piece "11100", "11001","10101",or "11010" ... can meet the requirement." We will leave this part to interested readers.
This theorem forms the basis of our new algorithm Alg-IG.

In[8], Quan introduced the concept of execution interference and uses it as the basis of Alg-EG. Now we generalize this concept here to make it fit into our new approach.

Definition 4. *Given a task set and the(m,k)-pattern for each task, the execution interference suffered by τ_{ik} from all higher priority tasks, denoted by I_{ik}, equals the total portions of the execution time of all mandatory jobs of higer priority tasks that fall inside $[(k-1)T_i + O_i, kT_i + O_i]$.*

The concept of *excecution interference of* τ_{ik} also forms the basis of our new algorithm to be discussed next. and the precise computation of I_{ik} is outlined in Algorithm 3.1 in the appendix section.

Next let's see how to determine the reasonable pattern with our new algorithm. Before doing that,we should point out the fact that for any loop pattern for task τ_i, the pattern of all jobs at invocation $j + nk_i(kn \geq 0)$is the same and equal to the pattern at j (j *is a given value*, $1 \leq j \leq k_i$), so all invocation of the jobs can be categorized into k_i positions,namely:$1 + nk_i,\ 2 + nk_i, ..., k_i + nk_i$. Since all job invocations at the same position has the same attribute(*mandatory* or *optinonal*), we call those position with mandatory jobs as *mandatory positions* and those with optional jobs as *optional position*. If any job at the mandatory positions miss the deadline we will say the task set miss the constraint. Since the partition of positions is equal to the partition of (m,k)-patterns, we should try to avoid partitioning those positions with at least one job miss the deadline as *mandatory positions*.

As we know, whether a job will miss the deadline is determined by how much execution interference I_{ik} it suffers from higher priority tasks.
We define $\hat{I}_{ij}$, for$1 \leq j \leq k_i$, as *maximum interference* that each job at invocation $j + nk_i$ suffers from higher priority tasks. next we will show how to precisely compute the *maximum interference* that job at invocation $j + nk_i$ suffers from higher priority tasks.

Then the mandatory jobs is selected according to the values of $maximum\ interference$ of jobs at each invocation of $j + nk_i (1 \leq j \leq k_i)$. The idea is that for a given $j (1 \leq j \leq k_i)$, the less the $maximum\ interference$ of job at invocations $j + nk_i$,the less the possibility that the jobs at invocations $j + nk_i$ will miss the deadline. So we should select those jobs with less $maximum\ interference$ at invocation $j + nk_i$ as mandatary while designate those with lager $maximum\ interference$ at invocation $j + nk_i$ as optional. This idea is implemented with the algorithm 1. The algorithm uses Alg-EG as a start point. The reason is that Alg-EG produces evenly distributed patterns with their WCIPs shifted away. This kind of pattern has both advantages and disadvantages. Since we usually schedule higher priority tasks first. For tasks that can be scheduled with evenly distributed pattern, we should try to do so because each mandatory instance of high priority tasks can be regarded as a $interference\ block$ to low priority tasks. By adopting evenly distributed pattern with their WCIPs shifted away,from the perspective of low priority jobs, the $interference\ blocks$ are made to *sparesely* distributed to the most extend and it improves the chance for low priority jobs to be schedulable. So evenly distributed pattern with WCIPs shifted away, if schedulable, should try to be used. However, for the cases that evenly distributed pattern doesn't work, we should adjust the pattern correspondingly.

The advantage of this method is that after determining the (m,k)-pattern, we already know whether it's schedulable or not.For example, if $\max\{\hat{I}_{ik} | k$ is the job invocations chosen as mandatory $\} > (D_i - C_i)$, then we know this task is not schedulable by our algorithm, otherwise it is schedulable.

Algorithm Alg_IG is quite effective in improving the schedulability of task sets with (m,k) constraints. We will give experimental results later to illustrate this.

4 Performance Evaluation

In this section we will compare the performance of the three algorithms described above by experiment results. In our experiments, we randomly generate some task sets with 4 or 5 tasks. The period of each task is randomly generated from a uniform distribution between 10 to 50, and the deadline of each task is assumed to equal to its period.The m_i and k_i values are also randomly selected, where k_i is uniformly distributed between 2 and 10,and m_i is uniformly distributed between 1 and k_i. We partition the utilization factor values into intervals of length 0.2. Then, the execution time of each task is randomly selected such that the utilization values of the resulting task sets are uniformly distributed within each interval. To reduce statistical

Algorithm 1 Algorithm 3.3.1 : Determine the proper regular (m,k)-pattern of given task set

Input: Task set $T = \{\tau_1, \tau_2, ..., \tau_n\}$, where $\tau_i = \{O_i, T_i, D_i, C_i, m_i, k_i\}$

Output: whether the task set is schedulable

TRUE: schedulable with $\pi_{i1}, \cdots, \pi_{ij}$ //(m,k)-pattern of T

FALSE: not schedulable

for i from 1 to n **do**

  determine (m,k)-pattern of τ_i according to (7) ;

  if τ_i is not schedulable **then**

    MandatoryJobs = 0;

    for j from k_i to 1 **do**

      $\hat{I}_{ij}$=0;

      while $S_i(j + nk_i) \leq 2 * h_i + max\{O_i\}$ **do**

        I_{ij} is calculated according to Algorithm 7.1;

        if $\hat{I}_{ij} < I_{ij}$ **then**

          $\hat{I}_{ij} = I_{ij}$;

        end if

      end while

      if $\hat{I}_{ij} < (D_j - C_j)$ AND MandatoryJobs $< m_i$ **then**

        $\pi_{ij} = 1$;

      else

        $\pi_{ij} = 0$;

      end if

    end for

  end if

end for

errors, the number of task sets schedulable by at least one of the approaches is no less than 50 within each interval, or at least 5000 different task sets have been generated for the interval.

	No. of Schedulable Task Sets			**Improvement(%)**	
Utilization	**Alg_WH**	**Alg_EG**	**Alg_IG**	**IG - WH**	**IG - EG**
0.6 - 0.8	11	11	11	0	0
0.8 - 1.0	17	29	29	70.5	0
1.0 - 1.2	4	65	82	1500	26.1
1.2 - 1.4	0	64	112	NaN	75
1.4 - 1.6	0	3	13	NaN	333.3
1.6 - 1.8	0	0	2	NaN	NaN
1.8 - 2.0	0	0	0	NaN	NaN

Table 1: Experimental results comparing the five approaches

The experiment results are collected in Table 1. In our experiments, task sets with utilization valuses less than 0.6 are all hard schedulable, thus we simply discard them. None of the task set with utilization greater than 2.0 is schedulable with any of the approaches. From Table 1, one can conclude that Alg-EG performs better than Alg-WH whilst Alg_IG outperforms the other two. The improvement become more significant as the task set utilization factor values increase(especially between 1.0-1.6). In the experiments, as we expect, a task set is schedulable with Alg_IG as long as it is schedulable with Alg-EG.

5 Conclusion

In this project we compared the performance of two well known algorithms in scheduling soft real time systems with fixed priority. And by developing the methodologies in both algorithms, we derived a new off-line algorithms which combines the advantages of both algorithms and achieve better performance than both of them. All of the algorithms above are easy to implement and introduce low scheduler overhead. Simulation results indicate that the proposed method does significantly improved the overall shedulability of a real time system.

References

[1] G. Bernat. *Specification and analysis of Weakly Hard Real-Time Systems*. PhD thesis, Universsitat de les Illes Balears, Spain, 1997.

[2] Guillem Bernat, Alan Burns, and Albert Liamosi. Weakly hard real-time systems. *IEEE Transactions on Computers*, 50(4):308–321, April 2001.

[3] J.-Y. Chung, Jane W.S. Liu, and K.-J. Lin. Scheduling periodic jobs that allow imprecise results. *IEEE Transactions on Computers*, 39(9):1156–1175, Sep 1990.

[4] 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.

[5] G. Koren and D. Shasha. Skip-over: Algorithms and complexity for overloaded systems that allow skips. *Proceedings of Real-Time Systems Symposium*, pages 110–117, Dec 1995.

[6] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. *Proceedings of the 1989 IEEE Real-time System Symposium*, pages 166–171, 1989.

[7] 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.

[8] G. Quan and X. (Sharon) Hu. Enhanced fixed-priority scheduling with (m,k)-firm guarantee. *IEEE Real-Time Systems Symposium*, pages 79–88, 2000.

[9] Parameswaran Ramanathan. Overload management in real-time control applications using (m,k)-firm guarantee. *IEEE Transactions on Parallel and Distributed Systems*, 10(6):549–559, Jun 1999.