Analysis and Research of Two-Level Scheduling Profile for Open Real-Time System

Yongxian Jin, Jingzhou Huang

Abstract—In an open real-time system environment, the coexistence of different kinds of real-time and non-real-time applications makes the system scheduling mechanism face new requirements and challenges. One two-level scheduling scheme of the open real-time systems is introduced, and points out that hard and soft real-time applications are scheduled non-discriminately as the same type real-time applications, the Quality of Service (QoS) cannot be guaranteed. It has two flaws: The first, it can not differentiate scheduling priorities of hard and soft real-time applications, that is to say, it neglects characteristic differences between hard real-time applications and soft ones, so it does not suit a more complex real-time environment. The second, the worst case execution time of soft real-time applications cannot be predicted exactly, so it is not worth while to cost much spending in order to assure all soft real-time applications not to miss their deadlines, and doing that may cause resource wasting. In order to solve this problem, a novel two-level real-time scheduling mechanism (including scheduling profile and scheduling algorithm) which adds the process of dealing with soft real-time applications is proposed.

Finally, we verify real-time scheduling mechanism from two aspects of theory and experiment. The results indicate that our scheduling mechanism can achieve the following objectives. (1) It can reflect the difference of priority when scheduling hard and soft real-time applications. (2) It can ensure schedulability of hard real-time applications, that is, their rate of missing deadline is 0. (3) The overall rate of missing deadline of soft real-time applications can be less than 1. (4) The deadline of a non-real-time application is not set, whereas the scheduling algorithm that server \( S_1 \) uses can avoid the “starvation” of jobs and increase QOS. By doing that, our scheduling mechanism is more compatible with different types of applications and it will be applied more widely.

Keywords—Hard real-time; two-level scheduling profile; open real-time system; non-distinctive schedule; soft real-time.

I. INTRODUCTION

With the development of real-time systems, the application of open real-time system, in which hard, soft and non-real-time applications are included, is getting more and more widespread. This brings new demands and challenges to the scheduling. So those scheduling approaches, which are proposed for closed real-time systems and suitable for simplex scope, have already not meet people’s demands. Therefore the concept “open real-time system” (ORTS) has been proposed in recent years. The ORTS’s uppermost characteristic is: when the system is running, all the real-time or non-real-time applications that are developed and validated independently can be configured dynamically and join in the system, then they will concurrent with original applications; and at the time the system is expanded dynamically, the global schedulability analysis is not needed [1].

After studying scheduling profile [1], this paper points out the scheme that schedules hard and soft real-time applications by regarding them as the same ones cannot assure QoS, then an improved scheme is put forward. The improved scheme adds the function of dealing with soft real-time applications, so the problem that schedules hard and soft real-time applications nondiscriminately is solved. Meanwhile we have taken account of influences to schedulability brought by non-preemptable sections (NPS). By doing that, our scheduling mechanism is more compatible with different types of applications and it will be applied more widely. The rest of this paper is organized as follows. Section II describes related works. In section III, we analyze the limitations of two-level scheduling profile. Section IV presents a novel scheduling profile and scheduling algorithm. Section V and VI give the schedulability analysis and simulation results. We conclude with a short summary and extensions on future work in Section VII.

II. RELATED WORKS

At present, scheduling profile of ORTS include two kinds: the first is the method integrating a variety of scheduling algorithms based on servers within the hierarchical scheduling framework [1]. It is one of the bandwidth reservation algorithms using CUS (constant utilization server) and TBS (total bandwidth server). The two-level scheduling profile is established based on that. It focuses on individualized task scheduling to the application system; the second is the method syncretizing a variety of scheduling algorithms within a unified architecture [2]. It employs a unified system-scheduling model which contains some different scheduling strategies. It permits to configure multiple scheduling strategies in a unified structure, but the system can use only one strategy when running. On the basis of the two kinds of scheduling profile, researchers have brought forward lots of scheduling algorithms for kinds of scheduling objects existing simultaneity in an ORTS.

Generalized Processor Sharing (GPS) algorithm [3] idealizes real-time applications to be a work-flow whose granularity can be subdivided infinitely, and then each real-time task will be allocated certain CPU bandwidth according to its demand. EGPS algorithm [4] inherits the thought in [3]. Constant Bandwidth Server (CBS) [5,6] and Hierarchical CBS (H-CBS) algorithm [7] focus on the problem of providing efficient run-time support to multimedia applications in a real-time system, where different types of tasks can coexist. The bandwidth

Yong-xian JIN is with College of Mathematics, Physics and Information Science, Zhejiang Normal University, China (E-mail: jyx@zjnu.cn)
reservation mechanism allows real-time tasks to execute in a dynamic environment under a temporal protection mechanism, so that each task will never exceed a predefined bandwidth, independently of its actual request. Earliest Deadline as Late as possible-Red Tasks Only (EDL-RTO) and Blue When Possible (EDL-BWP) [8] are two on-line algorithms, and the objective is to minimize the average response time of soft aperiodic request, while ensuring that the QoS (Quality of Service) of periodic tasks will never be less than a specified bound. Reference [9] presents the non-preemptive Group-EDF algorithm for soft multimedia-application systems. The experiment suggests this algorithm is more efficient in executing soft multimedia applications. Reference [10] focuses on scheduling soft real-time applications of the multiprocessor platform. It points out that compared with partitioned EDF, global EDF can get a higher system utilization. Processor Sharing with Earliest Deadlines First (PShED) algorithm [11] provides independence among scheduling tasks. Rigorously Proportional Dispatching Server (RPDS) algorithm [12] establishes a new hierarchical scheduling framework, and it schedules types of tasks using time chip as the basic unit. Open Adaptive Real-Time Scheduling (OARTS) framework [13], which imports auto control ideas into ORTS scheduling, can adjust real-time priorities of tasks depending on local resources. However, it do not think about characteristics of tasks such as NPS, global resources etc. And they increase the burden of the system because of a mass of computing. Reference [14] presents Two-Dimensional Priority Real-Time Scheduling (TDPRTS), and the system allocates different priorities and corresponding bandwidth for different algorithms, but the bandwidth can not be adjusted dynamically. The mechanism is not agile enough so that it is difficult to make full use of system resources. Reference [15] presents a multiprocessor scheduling framework for integrating hard and soft and best-effort (non real-time) tasks. It ensures that hard real-time deadlines are met and that of soft ones are less than a bound.

Research above does not synthetically think about the type of tasks (periodic or aperiodic), the characteristic of tasks (if they contain NPS, if they require global resources) and so on. By contrast, the two-level scheduling profile [1] has its own advantage. The reason is that it admits real-time and non-real-time applications and tasks with different characteristics, and it can schedule real-time and non-real-time applications in a complex open real-time environment.

III. ANALYSIS OF THE TWO-LEVEL SCHEDULING PROFILE

A. Related Concepts

Definition 1 Open Real-Time System: Non-relevant real-time applications and non-real time applications may be developed and validated independently, and global schedulability analysis is not necessary when the system is extended dynamically [1].

Definition 2 Task: Software entity that can accomplish some function. It is a basic unit of real-time scheduling. An execution during the task’s lifetime is called a job of this task. Application is defined by a set consists of multiple tasks.

Definition 3 Server: In this paper it represents a special task established by the system scheduling mechanism. It provides service for scheduling objects [16]. There is more than one server in system and each server is equivalent to a slow processor.

Definition 4 Server Speed: Assume that the speed of system processor is 1. Consider a server as a virtual processor, and then the speed ratio of virtual processor and the system processor is server speed. The server speed of $S_k$ is: $\sigma_k < 1$.

Definition 5 Application Event: It means an action occurred inside the application at some point. In order to ensure schedulability, the server scheduler must keep track of moments application events take place. An event of $A_k$ refers to one of the following: (1) a job in $A_k$ is released or completes; (2) a job in $A_k$ requests for or releases a global resource; (3) a job in $A_k$ enters or leaves a NPS.

B. Analysis for Two-level Scheduling Profile

Fig. 1 shows the hierarchical (or called two-level) scheduling framework [1].

The workload of the processor consists of a variable number $N$ of real-time applications, called $A_1, A_2, \ldots, A_N$, together with non-real-time applications. All non-real-time applications are executed by a server $S_0$, while each real-time application is executed by a server $S_k$ ($k \geq 1$). The $S_0$, $S_1, \ldots, S_N$ are at the upper level. Each server $S_k$ has a
ready queue containing ready jobs of the real-time application \( A_i \), and ready queue of the server \( S_0 \) contains ready jobs of all the non-real-time applications. The server \( S_0 \) uses a time-sharing algorithm to schedule ready jobs of all non-real-time applications [17]. At the OS level, the scheduler provided by the operating system, which we call OS scheduler, maintains all the servers in the system. It replenishes the server budget and sets the server deadline for every server according to the characteristics of the applications. A server is ready when its budget is nonzero and its ready queue is not empty. The OS scheduler also has a ready queue, which contains all the ready servers. It schedules all the ready servers according to the EDF algorithm. The hierarchical scheduling scheme, which is based on two kinds of servers CUS (Constant Utilization Server) and TBS (Total Bandwidth Server), is one kind of server scheduling strategies. Server \( S_j \) may be CUS or TBS, and it must participate in the system scheduling with deadline. The difference of CUS and TBS is following:

Assume that \( S_j \) is a CUS. If its deadline is \( d_{i,k} \) at time \( t \),

\[
d_{i,k} = \max \{ t_i, d_{i,k-1} \} + e_{i,k} / \sigma_{i,k}
\]

Here \( \sigma_{i,k} \) denotes the server speed of \( S_j \), and \( e_{i,k} \) denotes the remaining WCET (worst case execution time). Server \( S_j \) replenishes its budget at time \( \max \{ t_i, d_{i,k-1} \} \), and the budget value is \( e_{i,k} \). The deadline setting of TBS is the same as CUS, and the difference between them is replenishment time. Suppose job \( J_{i,k} \) is released at time \( t \): if \( t \geq d_{i,k-1} \), they are the same; if \( t < d_{i,k-1} \), TBS can replenish its budget as soon as the job \( J_{i,k} \) is finished, while CUS has to wait till to time \( d_{i,k-1} \)[16].

Scheduling framework mentioned above can manage real-time applications and non-real-time applications. About scheduling non-real-time applications, we have already discussed in detail in [18]. About real-time applications scheduling, the original framework has some characteristics as below:

(1) Each real-time application has its own scheduling algorithm and server. Each server is allocated certain bandwidth and it must maintain the current application’s ready queue and take part in scheduling in OS level.

(2) There are no restrictions on the type of tasks contained in real-time applications. Periodic tasks, aperiodic tasks and sporadic tasks are all allowed.

(3) It does not strictly limit tasks’ characteristics in a real-time application: 1) tasks may preempt each other. 2) Tasks may request for global resources. 3) Tasks may have NPS. For those tasks that are nonpredictable or request for global resources or have nonpreemptable sections, we must prepare sufficient bandwidth in advance to ensure their schedulability.

(4) When a real-time application enters the system, global schedulability analysis is not necessary, but the application has to pass the acceptance test. Only applications that meet some conditions can be accepted.

From the four points we know that the original framework can manage multi-types real-time tasks with a variety of characteristics and ensure their schedulability. However, it does not consider the differences between hard and soft real-time applications when scheduling them, that is to say, it applies only to hard-real-time applications. If \( T_j \) is a hard real-time task, \( e_j \) represents its job’s WCET without being interrupted. If \( T_j \) is a soft real-time task, \( e_j \) is a statistical or estimated value [12]. So for a soft real-time application which is made up of soft real-time tasks, the server speed and execution time it demands are estimated values when executing. If we use the original framework to execute a soft real-time application and its actual execution time is longer than the estimated value, we will not ensure the schedulability of system. In addition, there is another difference between hard and soft real-time applications. A hard real-time application must not miss its deadline; otherwise it will result in a disaster. A soft real-time application has an estimated deadline, even though some miss their deadlines, it will not have a significant loss. However, in practice, we should try out best to lower the rate of missing deadline to increase QOS (Quality of Service).

For that reason, we attempt to modify the part of scheduling real-time applications. It is hoped to present a new scheduling mechanism to achieve the following objectives:

(1) It should reflect the difference of priority when scheduling hard and soft real-time applications.

(2) It should ensure schedulability of hard real-time applications, that is, their rate of missing deadline is still 0.

(3) The overall rate of missing deadline of soft real-time applications should be less than 1.

(4) The deadline of a non-real-time application is not set, whereas the scheduling algorithm server \( S_j \) uses should try to avoid the “starvation” of jobs.

IV. A NOVEL SCHEDULING PROFILE AND SCHEDULING ALGORITHM

To achieve our objectives, a novel scheduling profile is as follows (shown in Fig. 2). The server ready queue of OS level is divided into three parts: hard, soft and non-real-time respectively. Server schedulers of hard and soft real-time applications use the EDF algorithm, and that of non-real-time applications uses a time-sharing algorithm.

The discussion will be carried on the premise of the following conditions:

(1) Tasks in hard and soft real-time applications are all periodic, and they are scheduled by preemptive algorithms.

(2) All applications are predictable. Periods of tasks in applications are equal to their relative deadlines, and a job is always released at the beginning of the period.

(3) Hard and soft real-time applications can include NPS or use global resources, but if a non-real-time application includes NPS or use global resources, it will be scheduled as a soft real-time application.

International Scholarly and Scientific Research & Innovation 3(3) 2009 774 scholar.waset.org/1999.4/6115
(4) In this paper a task is the basic scheduling unit. An application’s schedulability is proved through the schedulability of tasks it includes, that is, if tasks in an application are all schedulable, the application is schedulable.

![Diagram of scheduling profile]

The general idea of our new scheduling algorithm is that different priorities are given three server ready queues which respectively have hard real-time jobs, soft real-time jobs and non-real-time jobs. In the system if there are hard real-time jobs which are ready, they will be executed firstly. When the server ready queue of hard real-time jobs is empty, soft real-time jobs will be executed. When the two ready queues above are empty, non-real-time jobs will be executed. When a job with lower priority is being executed, if a job with higher priority becomes ready, it will preempt the former one immediately. In the following text, some parameters are:

- $U_h$: The fixed bandwidth allocated to non-real-time applications when the system starts. It does not change till the system terminates.
- $U_s$: The total CPU utilization ratio of all hard real-time applications in the system.
- $U_i$: The total CPU utilization ratio of all soft real-time applications in the system.
- $U_f$: The total CPU utilization ratio of system, namely the total bandwidth already allocated, it meets the equation:

$$U_f = U_h + U_i + U_0$$  \hspace{1cm} (2)

and

$$U_i \leq 1$$  \hspace{1cm} (3)

- $B_j$: The maximum execution time of the NPS of all jobs other than $J_f$, that is, $B_j$ is the maximum blocking time the job $J_f$ can suffer due to the non-preemptivity of other jobs.

An application must accept the acceptance test before entrance. If a non-real-time application contains NPS or needs global resources, it will be scheduled as a soft real-time application. Otherwise it will directly enter the scheduling queue of non-real-time applications. When a new application $A_k$ requests for admittance, it provides the following information in its admission request:

- The scheduling algorithm $\sum_k$ by $A_k$.
- The speed $\sigma_k$ of the slow processor on which $A_k$ is schedulable.
- The maximum length $L_k$ of all NPS or critical sections guarding global resources used by $A_k$.
- The shortest relative deadline $\delta_j$ of all jobs in $A_k$ if $A_k$ is a priority driven application, or the shortest length $\delta_j$ between any two consecutive events of $A_k$ if $A_k$ is a time-driven application.

Then the system will respond to that as follows:

- Find the type of the server for $A_k$, TBS or CUS (in this paper it must be TBS).
- Allocate the bandwidth $U_i$ for $A_k$ (here, application are predictable, so $U_i = \sigma_k$).
- If $U_i + U_f + \max_{j,k} \{B_j / \delta_j \} > 1$ ($N$ is total number of applications in the system including $A_k$), reject $A_k$. Else, admit $A_k$, and do follow work: create a TBS $S_k$ with server speed $U_i$ for $A_k$; set server budget and server deadline to zero and increase $U_i$ by $U_i$ [1].

At application level, if an application releases a new job, the job will be inserted to the ready queue in order according to the algorithm used by the scheduler. At OS level, the system must check ready queues of all real-time applications continuously. When a job is ready, the system replenishes the corresponding server’s budget and sets its deadline. Then the server will join the corresponding server ready queue. The server scheduling algorithm is:

**Step1:** Check whether the hard real-time server ready queue is empty, if empty, go to Step2; else, schedule the first server in the ready queue and execute the job. When the execution completes, move the server out of the server ready queue, go to Step1.

**Step2:** Check whether the soft real-time server ready queue is empty, if empty, go to Step3; else, schedule the first server in the ready queue and execute the job. When the execution completes, move the server out of the server ready queue, go to Step1.

**Step3:** Check whether the non-real-time server is ready, if it is ready, execute jobs till the moment the next application event occurs; go to Step1.

**IV. Schedulability Proof**

Next we will verify whether our two-level scheduling profile can achieve the anticipatory objects:

1. Through the description of the algorithm, we know that hard real-time jobs always have priority over soft real-time and non real-time jobs, and hard real-time jobs are blocked only happens when soft ones are in their NPS or holding...
The two jobs they release are respectively \( J_1 \) and \( J_2 \). The time interval of NPS in \( J_1 \) is \( t_1 - t_0 \), and its execution time is \( e_1 \). Its release time \( r_1 \) is after \( t_0 \) while its deadline \( d_1 \) is right before \( d_2 \), and it makes \( J_2 \) unschedulable. From 1) we know \( J_1 \) is schedulable. According to the relationship between \( p_1 \) and \( p_2 \) (respectively the period of \( J_1 \) and \( J_2 \) ), we will discuss the following two cases:

(i) \( p_1 \leq p_2 \)

From the condition above, we get \( e_1 / p_2 \leq e_1 / p_1 \). And from Fig. 3, we know that whenever \( J_i \) is released, the following inequation (6) is always true.

\[
\frac{t_n}{p_1} + e_2 > P_2
\]

So:

\[
1 < t_n / p_2 + e_1 / p_2 + \sigma_2
\]

Furthermore:

\[
1 \leq t_n / p_2 + e_1 / p_2 + \sigma_2
\]

\[
= t_n / p_2 + \sigma_2
\]

\[
< U_i + \max_{i \in J} \{ B_i / \delta_i \}
\]

It is in contradiction with the permission condition. The assumption is wrong.

(ii) \( p_1 > p_2 \)

According to the assumption, we have \( t_0 < r_1 < r_2 \), then

\[
(t_1 - t_0) / p_2 + e_1 / p_2 + e_1 / p_2 \geq (t_1 - r_1) / p_2 + e_2 / p_2 + e_1 / p_2
\]

\[
= (t_1 - r_1) / p_2 + (e_2 + e_1) / p_2
\]

\[
= (t_1 - r_1 + d_1 - t_1) / p_1
\]

\[
= 1
\]

On the other hand,

\[
(t_1 - t_0) / p_2 + e_1 / p_2 + e_1 / p_2 \leq \max_{i \in J} \{ B_i / \delta_i \} + \sigma_2 + \sigma_1
\]

\[
< \max_{i \in J} \{ B_i / \delta_i \} + U_i
\]

So:

\[
1 < (t_1 - t_0) / p_1 + e_1 / p_1 + e_2 / p_2
\]

\[
< \max_{i \in J} \{ B_i / \delta_i \} + U_i
\]

That is:

\[
1 < \max_{i \in J} \{ B_i / \delta_i \} + U_i
\]

It is in contradiction with the permission condition. The assumption is wrong.
From (i) and (ii) we know $J_i$ will not affect $J_2$’s schedulability. Other cases can be proved like that.

1) and 2) tell us that whether directly or indirectly, a hard real-time job which has NPS or use global resources will not affect the schedulability of the other hard ones. So the schedulability of a hard real-time application made up of hard real-time tasks can be ensured.

**Theorem 3:** if the system uses the EDF algorithm to schedule the server ready queue of hard real-time applications, then NPS contained in a soft real-time application or global resource will not affect the schedulability of hard ones.

**Proof:** here, $\max_{i,j} \left\{ B_i \right\}$ applies to all the hard and soft real-time applications. When a job enters its NPS, regardless of hard one or soft one, its affection to the schedulability of a hard real-time job is all the same. So the proof is the same as that of Theorem 2.

From the three theorems above we see that our scheme and algorithm can ensure hard real-time applications’ schedulability.

(3) Next we will validate whether soft real-time applications’ total rate of missing deadline is less than 1 in the improved scheme.

If there are no hard real-time applications, soft real-time applications will get the highest priority. Theorem 1 and Theorem 2 show that they are schedulable statistically. Therefore, the existence of hard real-time applications is the reason why the soft ones miss their deadlines. Assume there are $m$ hard real-time tasks and $n$ soft ones, and the lowest common multiple of all tasks’ periods is $p$. If every execution of these $n$ soft real-time tasks has missed its deadline, then according to the algorithm, it is impossible to execute non-real-time tasks within $p$. That’s to say, during the time interval, the total CPU utilization ratio it needs which makes all tasks schedulable is more than 1. However, all tasks are periodic so that the state will not be improved. It contradicts the permission condition. Consequently at least one soft real-time job must complete within $p$. Then it is concluded that the statistic rate of missing deadline of soft real-time applications is less than 1.

(4) When it starts the system reserves bandwidth $U_0$ for non-real-time applications. The two-level profile can implement the bandwidth isolation, so within $p$, non-real-time jobs will execute for $p \times U_0$. It has been discussed in [18] and that approach can avoid “starvation” and increase QOS.

VI. SIMULATION AND ANALYSIS

Following we make a simulation to test the validity of the new scheduling algorithm.

A. Testing Environment

This experiment is performed in a PC with single CPU, and main parameters are:

1) CPU: Pentium4 frequency 3.0G Hz;

2) Main memory: DDR SDRAM, frequency 400MHz,

3) External storage: Western Digital, rev 7200 t/m, capacity 80G;

4) Operating system: WINDOWS XP PROFESSIONAL;

5) Programming language: JAVA 2, JDK 1.6.0;

6) Programming environment: ECLIPSE 3.4.1;


B. schematic design

In order to highlight the algorithm’s performance and eliminate influences from testing environment, scheduling spending and timer error to results, periods of real-time tasks are set in second grade. Tasks’ characteristic parameters, such as the number of tasks, if a task contains NPS, the length of NPS, a task is hard one or soft one, the length of a task’s period and the bandwidth a task needs, are generated automatically by system. In the process 30 tasks are generated, and each task is executed 10 times, and periods of tasks range from 10s to 60s. The result is counted according to the respective execution number of hard and soft real-time tasks, specifically, the respective DMR (Deadline Miss Ratio) is computed every 30 executions. The testing program is a multithreading one, and it is made up of four threads, which are:

1) Task generation thread: It is responsible for task generation. When generating tasks, it sets the length of a task, the bandwidth a task needs, WCET, the execution number and determines whether a real-time task is hard or soft. New tasks join the waiting queue. The thread will wait when the waiting queue is full.

2) Acceptance test thread: This thread examines the task generation queue and performs acceptance test to tasks in the queue. Tasks that meet conditions enter the system, and then they are removed from the queue. When the system can not accept any task in the queue, the thread notifies task generation thread to generate new tasks. After that the thread waits till some task quits from system.

3) Task execution thread: This thread executes and switches real-time tasks in system according to the algorithm described in 3.3.2. When some task is running, if another task is ready, the system will stop executing the current task, save its context state and put it into the corresponding ready queue in order. Then the system will schedule real-time tasks in the ready queue over again (only one exception: if the current real-time task is a hard one and a soft one is ready, the execution will not be interrupted). If the current task is located in its NPS, the system will not act until the task leaves its NPS.

4) Waiting queue of system tasks maintenance thread: Tasks executed by system are periodic, so after one execution a task has to wait till the next period begins, and at this moment it will be inserted into the waiting queue of system tasks. The system examines tasks in the queue. When some task’s period begins, it will be put into the corresponding ready queue.

C. Results Analysis
We make experiments for different load efficiency, and hard and soft real-time tasks execute more than 250 times respectively.

(1) Light load: As is shown in Fig.4, we can see DMR of all hard and soft real-time tasks stay 0 all the time (the two curved shapes coincide with X axis), in other words, there are no tasks that miss their deadlines. The result is in accord with our supposition.

(2) Middle load: Through experiment we know all tasks meet their deadlines. The algorithm we use has the effects as good as that on light load condition. So we get the graph just the same as Fig. 4.

(3) Full load: According to simulation result (shown in Fig. 5), no hard real-time tasks miss their deadlines (the curved shape coincides with X axis), while the maximum of missing deadline rate of soft real-time tasks is smaller than 7%. The result testifies theorems and indicates our scheduling algorithm can achieve the anticipatory objects. In addition, perhaps soft real-time tasks can not complete before their deadlines, but statistically the system provides better concurrency for them. The system can accept more soft real-time tasks and execute them concurrently if only the NPS contained in them do not affect the schedulability of hard ones. Thus the system can stay full load evermore and make full use of resources.

(4) Over load: The system does not deny any task, it means the load may vary in a big range. Therefore, it is not proper to display all results in a graph, but we can analyze them. When $U_n$ is bigger than 1, CPU will execute hard real-time tasks all the time, while soft real-time tasks will not be executed. This is not an extreme situation. Now QOS of soft real-time tasks is 0, and we can not conclude accurately how many hard real-time tasks will miss their deadlines, but some of them will have to. The framework and corresponding algorithm is not suitable for the over load condition.
VII. CONCLUSIONS AND FUTURE WORK

This paper focuses on the two-level scheduling profile of ORTS and points out the limitation that the profile only suits a hard real-time environment, and then we present an improved scheduling approach which adds the process of dealing with soft real-time applications. The improved approach solves the problem that schedules hard and soft real-time applications nondistinctively, and it can be applied in a more complex environment. Through proof and simulation, we can see the improved approach can achieve the prospective objectives.

It should be pointed out that in more complex environments, an application is not predictable or it includes sporadic tasks, this paper does not discuss. Those problems need to be solved in the future.

REFERENCES


