International Journal of Computer Theory and Engineering, Vol. 1, No.2,June2009
1793-8201

Scheduling for Non-Real Time Applications of ORTS Based on Two-Level Scheduling Scheme

YongXian JIN and JingZhou Huang

***Abstract*—In an open real-time system, 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 was introduced. Through analysis, we find that the scheduling strategy for non real-time applications in original two-level scheduling scheme is too simple: it may make real-time applications unschedulable if non real-time applications contain non-preemptive sections (NPS). In order to avoid that, this paper proposes four pointed scheduling rules. Then by integrating the improved scheduling algorithm for non-real-time applications, we can solve problems existing in non real-time applications scheduling. Ultimately, the schedulability of real-time applications and non-real time applications can be guaranteed.**

***Index Terms*—Non-preemptable section (NPS), non real-time application, open real-time system (ORTS), schedulability, two-level scheduling scheme.**

I. Introduction

With the development of information technology, application of computer systems is getting more and more widespread. Growing requirement of dealing with real-time information makes the increasingly close relationship between real-time systems and people. Early real-time system, which has single type of task, scheduling approach and scheduling object, and tasks cannot join or withdraw from the system dynamically, is named closed real-time system. To this day, that kind of closed real-time system has been unable to meet the people's needs, while the corresponding one, named open real-time system (ORTS), has become more and more popular. The ORTS's uppermost characteristic is openness, and it has two aspects. The first is openness of application types. In ORTS, multi-types applications including hard real-time applications, soft real-time applications, and non-real time applications, may be concurrent at the same time. The second is openness during runtime. During system's runtime, kinds of applications may join or withdraw from the system dynamically according to some conditions. So those primary scheduling approaches, which are proposed for closed real-time system and suitable for simplex scope, have already not meet people's demand.

This paper focuses on the two-level scheme [1], through analysis, we find that the scheduling strategy for non-real-time applications in original two-level scheduling scheme is too simple: it may make real-time applications unschedulable if non-real-time applications contain non-preemptive sections (NPS). In order to avoid that, this paper proposes four pointed scheduling rules and scheduling algorithm for non-real-time applications that can perfect it. Finally, feasibility is analysed

The rest of this paper is organized as follows. Section II describes related works. In section III, we analyse the present two-level scheduling scheme of open real-time system. Section IV analyses scheduling of real-time and non-real time applications in two-level scheduling scheme. Section V gives existing problems and solving approaches of non-real time applications scheduling of two-level scheduling scheme for ORTS. We conclude with a short summary in Section VI.

Yong-xian JIN is with College of Mathematics, Physics and Information Science, Zhejiang Normal University, China.

II. Related Works

At present, scheduling mechanism 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 framework 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 mechanism, researchers have brought forward lots of scheduling algorithms for kinds of scheduling objects existing simultaneity in an ORTS.

GPS（Generalized Processor Sharing） 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]. CBS（Constant Bandwidth Server）[5, 6] and H-CBS（Hierarchical CBS） algorithm[7] focuses 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 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. EDL-RTO

（Earliest Deadline as Late as possible-Red Tasks Only）and EDL-BWP（Blue When Possible）[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. PShED（Processor Sharing with Earliest Deadlines First）algorithm [11] provides independence among scheduling tasks. RPDS（Rigorously Proportional Dispatching Server） algorithm [12] establishes a new hierarchical scheduling framework, and it schedules types of tasks using time chip as the basic unit. OARTS（Open Adaptive Real-Time Scheduling）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 burden of the system because of a mass of computing. Reference [14] presents TDPRTS (Two-Dimensional Priority Real-Time Scheduling), 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 scheme [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. Two-Level Scheduling Scheme of The ORTS

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 presents a special task established by the system's scheduling mechanism and 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: It is assumed that the speed of system processor is 1. Consider one server as a virtual processor, and then the ratio of the speed of virtual processor and the system processor is Server Speed. The server speed of S_k is: $s_k < 1$.

B. Two-level scheduling scheme

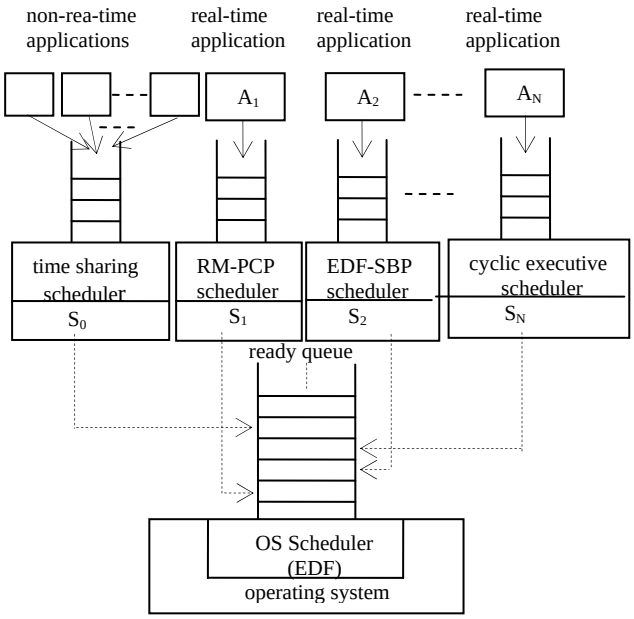

Fig. 1 Two-level scheduling architecture of open real-time system

Fig.1 shows the architecture of ORTS supported by the two-level hierarchical scheduling scheme. The system has a single processor whose speed is one. The workload of the processor consists of a variable number N of real-time applications, called A_1, A_2, …, 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 servers S_0, S_1, …, S_N are at the upper level (application level). Each server S_k ($k \geq 1$) has a ready queue containing ready jobs of the real-time application A_k, and ready queue of the server S_0 contains ready jobs of all the non real-time applications. The server scheduler of the server S_0 uses a time-sharing algorithm to schedule ready jobs of all non real-time applications in order to ensure impartiality. At the lower level (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 the server executes. 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 in the system. It schedules all the ready servers according to the EDF algorithm.

Scheduling Of Real-Time And Non-Real Time Applications In Two-Level Scheduling Scheme

The two-level scheduling scheme, which is based on two kinds of servers called CUS (Constant Utilization Server) and TBS (Total Bandwidth Server), is one of server scheduling strategies. Server S_i shown in Fig.1 may be CUS or TBS, and it must participate in scheduling of the system with deadline. The following explanation is necessary:

Assume that S_i is a CUS server in ORTS. If its deadline is $d_{i,k}$ at time t, $d_{i,k}$ is calculated as follows:

$$d_{i,k} = \max\{t, d_{i,k-1}\} + e_{i,k} / s_{i,k} \quad (1)$$

Here $s_{i,k}$ denotes the speed of server S_i, and $e_{i,k}$ denotes the remaining WCET (worst case execution time). S_i replenishes its budget at time $\max\{t, d_{i,k-1}\}$, and the budget value is $e_{i,k}$ [16]. The deadline setting of TBS is the same to 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 $J_{i,k-1}$ is finished, while CUS has to wait till to time $d_{i,k-1}$ [17].

A. Scheduling of Real-Time Applications

Reference [17] proposed some schedulability conditions of real-time applications (including hard real-time applications and soft real-time applications) that do not have NPS or use global resources in an ORTS, however, the schedulability of all kinds of applications has not been guaranteed yet, because we have to think about the following problems:

(1) If applications include aperiodic or aporadic tasks
(2) If applications are predictable
(3) If applications are scheduled by a preemptive algorithm
(4) If applications include NPS or use global resources

Reference [1] solved the upper four problems for real-time applications, but it did not consider the case that non-real time applications may include NPS or use global resources. The latter two problems are mentioned in this paper, so the solutions to them are followed. Let's see an example related to Fig.2. The application A_k in the example uses the EDF algorithm to schedule its two jobs, $J_1(0,10,44)$ and $J_2(40-a,1,44-a)$, where $0<a<40$. The application A_k is schedulable if it executes alone on a slow processor with speed 0.25, as shown in Fig.2 (a). Now suppose that the release time of the jobs in A_k are known, the application A_k is executed by a CUS whose server speed is 0.25, as shown in Fig.2 (b). Everything goes well until job J_2 completes and server S_k is no longer ready to execute. Before the server becomes ready again at $44-a$, the processor executes a job J in another application whose server has a deadline later than 44, and the job J enters its NPS right before $44-a$. Server S_k cannot execute the remaining piece of job J_1 until the job J leaves its NPS. If the length of the NPS is longer than a, J_1 misses its deadline at 44. However, if the application A_k is executed by a TBS with server speed 0.25, it is schedulable, as shown in Fig.2 (c). When the job J_2 completes at time $44-a$, the budget of the server is replenished immediately and the deadline is set to 44. The server remains ready to execute at this time. Consequently, the job J of the application whose server deadline is after 44 cannot execute before the job J_1 completes [1].

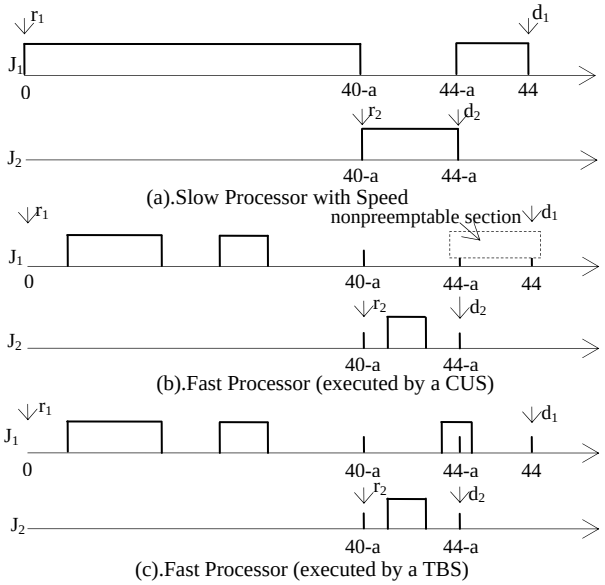

Fig.2 Schedules of real-time application A_k

Since the case in Fig.2 (b) may lead to un-schedulability of some applications, Reference [1] deals with that like this: If an application is scheduled by a preemptive algorithm when it enter the system, the system will check out if there have been applications including NPS. If any, the server type will be set to TBS; If not, the server type will be set to CUS. And the system will change CUSs scheduled by preemptive algorithms to TBSs if it accepts the first application including NPS.

B. Scheduling of Non-Real Time Applications

OS scheduler will establish a TBS S_0 firstly when the ORTS starts. The system accepts all the non-real time applications. Then they will be executed by S_0. The server S_0 is scheduled by time-sharing algorithm [18]. In reference [16] characteristics of CUS has been described. Suppose complete time of a job is t, and the server deadline is t' ($t<t'$). Then during the interval (t,t') (called background time) the processor is available, so S_0 can make use of this time interval to execute non real-time applications on it.

V. Existing Problems And Solving Scheme of Non-Real Time Applications Scheduling

A. Problems Description and Analysis

In front scheduling approaches for real-time applications and non-real time applications have been described, where description about non-real time applications scheduling is too simple. By analysis, the following problems are addressed:

The first problem: All the jobs released by non-real time applications are scheduled by time-sharing algorithm on server S_0 , and then can it guarantee the fairness and schedulability of executing all the jobs?

The second problem: The ORTS always admits non-real time applications, and non-real time applications do not provide any characteristic when they enter the system. Then can it affect the schedulability of the existent real-time applications?

The third problem: The system may execute non-real time applications by making use of background time of CUS; however, is it reasonable that non-real time applications are scheduled by current ways? And how does the server replenish its budget under the premise that the server is scheduled by time-sharing algorithm?

The analysis about the former three problems is as follows:

Analysis of the first Problem: Reference [1] uses a time-sharing algorithm in order to guarantee fairness of scheduling all the non-real time applications, so that jobs in the ready queue can be executed for the time divided equally from the budget. That's perfect, but perhaps the actual case is not so simple. Suppose:

(1) Every time if the server schedules job, and every job is executed by fixed time units, when there are amount of jobs the server may only execute some of jobs in the ready queue. If the number of jobs is so large that every time the server has to execute the latter jobs of the ready queue, then the former jobs in the ready queue will not be executed in a very long time. Then the case "starvation" appears. Since the case should not appear, the best solution is to increase the server speed. However, the server speed of S_0 is fixed when the system starts. Then how could we solve this problem? If all the jobs in the ready queue can obtain equal time units divided from budget every time the server is scheduled, every job will get very little time when there are lots of jobs. At this point the cost of context switching cannot be neglected, and how to solve this problem? Through the above description, disadvantages of the two time-sharing approaches can be seen clearly. What means should we use to avoid the two disadvantages?

(2) If NPS are included by jobs released by non-real time applications, the jobs may occupy the time that does not belong to them, so it will influence the following jobs' execution. If the task, which releases this kind of job, is periodic, it will destroy fairness of the time-sharing algorithm seriously. Then how could we deal with it? Assume a job which has NPS is executed at last in one scheduling, and at some point the job is been executing in its NPS, but the server budget runs out, then how should the server response?

Analysis of the second problem: in section III.A, there is an example describing the solution for potential unschedulability of real-time applications in ORTS. From the analysis two necessary conditions of unschedulability are obtained: one is application including NPS; the other is CUS scheduled by a preemptive algorithm. At first it limits the second condition, in other words, the type of server on which there are applications scheduled by preemptive algorithms, but preemption may occur all the same. And then it limits the first condition at the acceptance test, that is, $U_t + U_k + \max_{1 \le j \le N}\{B_j / d_j\} \le 1$, where $B_j = \max_{i \ne j}\{L_i\}$ and N is total number of applications in the system including A_k [1], so this problem is solved. Please refer to reference [1] to get more details. However, non-real time applications are not tested when they enter the system, so the system cannot get characteristics about NPS of them. Thus it is possible that the length of some non-preemptable section in non-real time applications is longer than that in real-time applications. So the potential unschedulability brought by the first necessary condition could not be avoided and schedulability conditions proposed by reference [1] could not be met. Then some real-time applications in the system may be not schedulable because of that.

Analysis of the third problem: The server S_0 may make full use of the background time of CUS if the system has some CUSs. However, perhaps the following cases appear:

(1) In a period of time, there are only TBSs in the system and the server speed U_0 of S_0 is small. If amount of jobs are in the ready queue, the case described in the first problem may appear. At that time the server S_0 can use spare bandwidth in the system temporarily. By doing this QOS (Quality of Service) of non-real time applications can be improved and the influence brought by inadequacy of bandwidth of the server S_0 can be eased too.

(2) How could it replenish the server budget when the budget runs out? We can do it like real-time applications: replenish the remaining WCET of the head job of the ready queue. However, the following problems may be found: the head job in the ready queue has been executed for many times and its WCET (assume it is T) is little. Suppose the server replenishes its budget with T . As mentioned above, if the second time-sharing way has been applied, the server will have divided its budget equally to all the jobs on it. Then every job can be executed for T / N , and at this time the WCET of the head job will change to $T - T / N$. If $N >> T$, then $T \approx T - T / N$. The expression shows us that all the jobs in the ready queue cannot be executed basically. At the same time the cost of context switching will not be neglected because it occupies a large proportion of executing time. Thus the server S_0 will participate in scheduling frequently with little budget. It increases burden of scheduling and even leads to unsteadiness of the system.

B. Scheduling Rules and Algorithm of *Non-Real Time Applications*

The section above addresses some problems that may

appear when all the non-real time applications are executed on S_0 of the TBS and analyses them in detail. These problems are brought by NPS of non-real time applications or using global resources, and inconsequence of scheduling non-real time applications.

It is existence of NPS or using global resources that destroys fairness and schedulability of scheduling non-real time applications. And it even results in some real-time applications unschedulable. This kind of application has its own particularity so it should not be scheduled together with other kinds of non-real time applications. The problem will be solved if these two kinds of applications are separated and non-real time applications including NPS or using global resources are scheduled together with real-time applications.

The second problem and the unschedulability problem of the first one are solved by making non-real time applications including NPS or using global resources join the real-time applications scheduling. Now concentrate on the remaining problems, namely: non-real time applications that do not have NPS or use global resources are scheduled on S_0 of a TBS, then how to avoid starvation of some jobs and improve QOS of other jobs as far as possible at the same time when the number of jobs in the ready queue increases? What kind of time strategy should the server use to schedule jobs? What way should the server use to replenish its budget? How to make use of the background time? Then the approaches are as follows:

Set two variables l and $l^{'}$ to denote bandwidth and one variable N to memorize the number of jobs in the ready queue of S_0. Initial value of $l^{'}$ is 0 and l is $1-U_0$(where U_0 is the server speed of S_0). When real-time applications enter or quit from the system, new value of l is calculated by this formula:

$$l = 1 - U_t - \max_{j \geq 1}\{B_j / d_j\} \quad (2)$$

Where, B_j is the execution time of the longest NPS of all the other applications other than A_k, and d_j is the shortest relative deadline of tasks in A_k, and U_t is the current total processor utilization of system. (i) If there is no background time, the server will replenish its budget with a fixed budget B and the deadline of it is set according to $U_0 + l^{'}$. (ii) If there is background time, the budget will be the length of the background time and the deadline is set to the deadline of the CUS server.

Suppose the shortest time unit that a job is executed each time is L. At first the server calculates the ratio between the current budget $B^{'}$ and the number N before it executes jobs. (i) If the inequation $L \leq B^{'} / N$ is true, then the budget will be divided equally for every job. If WCET of some jobs is less than the budget assigned, the remaining budget will be used to execute the next job. Keep doing this till the server budget runs out. If the budget is nonzero when all the jobs complete, OS scheduler will reclaim the budget. (ii) If the inequation $L > B^{'} / N$ is true, then L will be assigned to the front jobs of the ready queue. If some jobs have remaining budget, it will be dealt with like above.

If the number of jobs in the ready queue continues to increase and it is larger than a critical value N_M, the server will borrow the spare bandwidth temporarily to increase its speed till the number of jobs in the ready queue reduces to a value smaller than N_m, here, $N_M > N_m$ and there is a remainder between N_M and N_m. (i) If $N \leq N_m$, then set $l^{'} = 0$ and the server will participate in scheduling with its own server speed; (ii) If $N > N_M$, then set $l^{'} = l$ and the server will participate in scheduling with speed $U_0 + l^{'}$.

Compare l with $l^{'}$ if the value of l changes when S_0 is executing its jobs or in the ready queue: (i) If $l \geq l^{'}$, it will do nothing; (ii) If $l < l^{'}$, the server will stop executing or quit from the server ready queue immediately, make $l^{'} = l$ and set the remaining budget to new server budget. Then the server participates in scheduling over again.

According to analysis above, four scheduling rules are induced to resolve these problems addressed.

Scheduling Rule 1: acceptance rule of non-real time applications.

In the original framework non real-time applications join the scheduling queue of non real-time applications without accepting particular detections, while the our rules need to check if they contain NPSs and then deal with them depending on different circumstances.

All the non-real time applications must provide their parameter of NPS before they enter the system. If one application includes NPS or needs to use global resources, its priority will be improved and it will be tested and scheduled by the system as real-time applications; if not, the application will join in the server S_0. Following is the process of acceptance test (shown in Fig. 3).

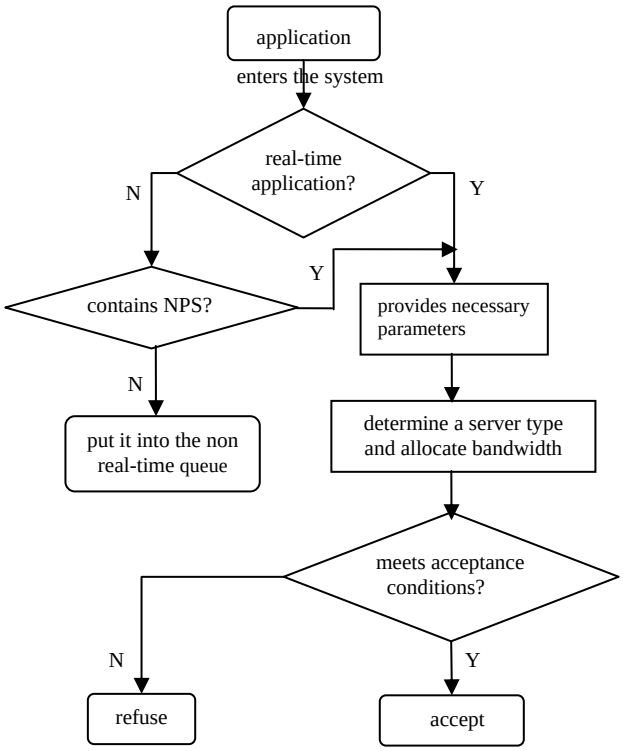

Fig.3 Acceptance test

(1)Check if the current application is a non real-time application. If it is, turn to (2), else to (3).

(2)Check if the current application contains NPS. If it does not, put it into the scheduling queue of non real-time applications, else turn to (3).

(3)The application provides necessary characteristic parameters for system, and then the system will determine a server type and allocate the corresponding bandwidth for it.

(4)Check if the total bandwidth allocated meets schedulable conditions. If it does, accept the application, else refuse it.

There is an issue to demonstrate. The system schedules hard, soft and non real-time applications depending on different priorities, but in the process of accept test, the system only differentiates real-time applications from non real-time ones. This indicates that there are no differences when the system chooses real-time applications, that is to say, hard real-time applications will not be chosen firstly just because they have higher priorities. It is the arrival order that determines if an application is chosen firstly, and fairness of acceptance test is reflected by doing this.

Scheduling Rule 2: replenishment rule of server budget.

When budget of the server S_0 is 0 and the ready queue is not empty: (1) If there is no background time, set the budget of the server to B and its relative deadline to $B/(U_0 + l^{'})$, then put it into the server ready queue of OS level;

(2) If there is background time from CUS, set budget of the server S_0 to the length of the background time and deadline to the deadline of the current CUS. And then the server executes jobs immediately.

Scheduling Rule 3: executing rule of jobs in the ready queue.

Suppose server budget is $B^{'}$ when the server is scheduled: 1) if the inequation $L \le B^{'}/N$ is true, then the server budget will be divided equally. Every job can be executed for $B^{'}/N$; 2) if the inequation $L > B^{'}/N$ is true, execute ready jobs circularly from the next job of the last one in the previous execution. Every job can be executed for L.

Scheduling Rule 4: borrowing rule of spare bandwidth.

(1) If $N \le N_m$, then set $l^{'} = 0$ and the server S_0 will participate in scheduling with its own server speed;

(2) If $N > N_M$, then set $l^{'} = l$ and the server S_0 will participate in scheduling with speed $U_0 + l^{'}$.

According to the rules mentioned above, one scheduling algorithm of non-real time applications is proposed as following:

(1) If the job ready queue of the server S_0 is empty, wait;

(2) If there is background time, set the budget of the server S_0 to the length of the background time and deadline to the deadline of the current CUS. Execute jobs in the ready queue according to rule 3, go to (5);

(3) Set the sever budget to B and execute rule 4;

(4) Execute rule 3. Compare l with $l^{'}$ if the value of l changes when S_0 is executing its jobs or in the ready queue: if $l < l^{'}$, the server will stop executing or quit from the server ready queue immediately and set the remaining budget to new server budget, execute rule 4, go to (4);

(5) If the budget of server S_0 runs out or the ready queue is empty then go to (1).

C. Validity Analysis of the Scheduling Scheme

In the four rules proposed: The first rule separates the non-real time applications including NPS or using global resources from other non-real time applications, and by improving priorities of them and making them participate in scheduling as real-time applications, the adverse effects brought by them are eliminated. Thus the problem addressed about non-real time applications including NPS or using global resources has been solved. Reference [1] has dealt with all kinds of real-time applications properly in an ORTS and prove the schedulability of them, so the scheme in this paper not only guarantees schedulability of non-real time applications including NPS or using global resources severely, but also guarantees this kind of application does not affect schedulability of other kinds of applications.

The three latter rules resolve the remaining problems addressed respectively. The time-sharing algorithm used by rule 3 guarantees execution fairness of all the jobs in the ready queue and eliminates the disadvantages of two primary time-sharing ways. Rule 4 makes full use of spare resources in the system when there are lots of jobs in the ready queue, so that it can improve the response speed and QOS of non-real time applications greatly. Setting about the server deadline in rule 2 guarantees that non-real time applications

will not occupy the bandwidth of real-time applications, that is to say, they will not affect real-time applications scheduling. So the scheme is feasible.

VI. Conclusion

By studying the scheduling approaches for the real-time and non-real time applications in the ORTS based on the two-level scheduling scheme, this paper points out limitations in non-real time applications scheduling, and then proposes one solving scheme. Finally feasibility is analysed. In conclusion, the schedulability of all applications in the system is guaranteed.

References

[1] Z.Deng, J.W.S. Liu. Scheduling Real-Time Applications in an Open Environment. In: Proc. of the 18th IEEE Real-Time Systems Symposium. IEEE Computer Society, 1997, pp.1-25.

[2] Y.C .Wang, K.J. Lin. Implementing a General Real-Time Scheduling Framework in the RED-Linux Real-Time Kernel. In: Proc. of the 20th IEEE Real-Time Systems Symposium. IEEE Computer Society, 1999, pp. 246-255.

[3] A.K. Parekh. A generalized processor sharing approach to flow control in integrated services networks [Ph.D. Thesis]. Massachusetts Institute of Technology, 1992.

[4] T.W. Kuo, W.R. Yang, K.J. Lin. EGPS: a class of real-time scheduling algorithms based on processor sharing. In: Proc. of the 10th Euromicro Workshop on Real Time Systems. IEEE Computer Society, 1998, pp.27-34.

[5] L.Abeni, G.Buttazzo. Integrating Multimedia Applications in Hard Real-Time Systems. In: Proc. of the 19th IEEE Real-Time Systems Symposium(RTSS'98). IEEE Computer Society, 1998, pp. 4-13.

[6] L.Abeni, G.Buttazzo. Resource Reservation in Dynamic Real-Time Systems. Real-Time Systems, 2004, 27: pp.123-167.

[7] G.Lipari, S.Baruah. A Hierarchical Extension to the Constant Bandwidth Server Framework. In: Proc. of the 7th IEEE Real Time Technology and Applications Symposium.IEEE Computer Society, 2001, pp. 26-35.

[8] A. Marchand, M. Silly-Chetto. Dynamic Real-time Scheduling of Firm Periodic Tasks with Hard and Soft Aperiodic Tasks. Real-Time Systems, 2006, 32(1-2): pp.21-47.

[9] W.Li, K.Kavi, R.Akl. A non-preemptive scheduling algorithm for soft real-time systems. Computers and Electrical Engineering, 2007, 33(1): pp.12-29.

[10] U.C. Devi, J.H.Anderson. Tardiness bounds under global EDF scheduling on a multiprocessor. Real-Time System, 2008, 38(2): pp.133-189.

[11] G.Lipai, J.Carpenter, S.Baruah. A Framework for Achieving Inter-Application Isolation in Multiprogrammed Hard Real-Time Environments. In: Proc. of the 21st IEEE Real-Time Systems Symposium. IEEE Computer Society, 2000, pp. 217-226.

[12] Y.C. Gong, L.G. Wang, *et al*. A Hybrid Real-Time Scheduling Algorithm Based on Rigorously Proportional Dispatching of Serving. Journal of Software, 2006, 17(3): pp. 611-619.(in Chinese)

[13] X.Y.Huai, Y.Zou, M.S.Li. An Open Adaptive Scheduling Algorithm for Open Hybrid Real-Time Systems. Journal of Software, 2004, 15(4): pp. 487-496. (in Chinese)

[14] P.L. Tan, H.Jin, M.H. Zhang. Two-Dimensional Priority Real-Time Scheduling for Open Systems. Acta Electronica Sinica, 2006, 34(1): pp. 1773-1777. (in Chinese)

[15] B. B. Brandenburg, J.H. Anderson. Integrating Hard/Soft Real-Time Tasks and Best-Effort Jobs on Multiprocessors. In: Proceedings of the 19th Euromicro Conference on Real-Time Systems. IEEE Computer Society, 2007, pp. 61-70.

[16] Y.Zou, M.S. Li, Q.Wang. Analysis for scheduling theory and approach of open real-time system. *Journal of Software*, 2003, 14(1):pp.83-90. (in Chinese)

[17] Deng.Z, Liu JWS, Sun J. A scheme for scheduling hard-real-time applications in open environment. In: *Proceedings of the 9th Euromicro Workshop on Real-Time Systems. Los Alamitos*, CA: IEEE Computer Society Press, 1997.pp.191-199.

[18] Deng. Z, Liu JWS, Sun J. Dynamic Scheduling of Hard Real-Time Applications in Open System Environment. *Technical Report UIUCDCS-R-96-1981, Department of Computer Science, University of Illinois at Urbana-Champaign*, 1996.