A Real-Time Task Scheduling Algorithm Based on Dynamic Priority

Hui Chen, Jiali Xia
School of Software, Jiangxi University of Finance and Economics, Nanchang, China

Abstract

By studying the dynamic value density and urgency of a task, a preemptive scheduling strategy based on dynamic priority assignment is proposed. In the strategy, two parameters \( p \) and \( q \) are used to adjust the weight that the value density and urgency of a task impact on its priority, and a parameter \( \beta \) is used to avoid the possible system thrashing. Finally, the simulations show that our algorithm is prior to the analogous algorithms, such as EDF, HVF and HVDF, on gained-value of the system, deadline miss ratio and preemptive number.

1 Introduction

Recently, the real-time system was widely applied in a number of such diverse environments as space or airborne platform management, factory process control, robotics and embedded intelligent devices [4, 2]. Usually, a real-time system is used to manage the tasks with the limitations of time and load, and its goal is to properly schedule the tasks and make sure as more tasks as possible to be accomplished before their deadlines. Thus, different priority assignment strategies [7, 4, 2, 3, 8, 6] have been proposed in the past few years.

Three attributes of a task, deadlines, slack times and value(density), are usually used to determine its priority. In order to make sure the tasks can catch their deadlines, the EDF [9, 7] method thinks that the task with earliest deadline should be executed first. However, the LSF [1] method assigns the highest priority to the task with least slack time, because it must be executed in order to catch its deadline. Additionally, in order to bring the maximal value to the system, the HVF [4] declares that the task with maximal value [10, 4] should be executed with first priority. Considering the executive efficiency, the HVDF [4] thinks that the task bringing maximal value per time-unit should be executed first.

However, it is not appropriate to assign the priority of a task just based on one of its attributes. For example, a task with nearest deadline might not always have biggest value, nor least slack time, and vice versa. If we stress effect of the deadline of a task on its priority, the urgent tasks could be executed first, but some tasks with higher value but less urgency might miss their deadline. Similarly, if a stronger emphasis is laid on the value of task, some urgent tasks might have no chance to obtain CPU, or be frequently preempted by the tasks with bigger values. In order to improve the holistic performance of a real-time system, synthetically analyze the time and value attributes of a task on its priority is necessary.

In this paper, we analyze the dynamic value density and urgency of a task according to its value, deadline and execution time, based on which, a dynamic priority assignment strategy and the scheduling algorithm (named DRTP) are proposed. Compared with the analogous algorithms, the DRTP algorithm firstly analyzes the dynamic value density of a task during its runtime at the first time. Secondly, both the value and time attributes of a task are considered when assigning its priority, and two parameter \( p \) and \( q \) are used to make a tradeoff between the effects of dynamic value density and urgency of the task on the priority. Thirdly, the DRTP algorithm discuss the possibility of system thrashing, and gives the condition that could avoid thrashing.

2 Transaction Model

Let \( T = \{T_1, T_2, \ldots, T_n\} \) be the task set in a real-time system, in which, each task \( T_i \) has the following attributes:

- \( C_i \), denotes the computation time,
- \( P_i \), denotes the period (or minimum inter-arrival time),
- \( D_i \), denotes the relative deadline,
- \( V_i \), denotes the expectant value brought by \( T_i \) to the system as \( T_i \) is successfully submitted,
- \( b_i \), denotes the arrive time,
- \( e_i \), denotes the finished time,
- \( d_i \), denotes the obsolete time, and \( d_i = b_i + D_i \),
- \( V \), denotes the actual value that \( T_i \) brings to the system, and \( v = \begin{cases} v_i & \text{if } e_i \leq d_i \\ 0 & \text{otherwise} \end{cases} \)

At any time \( \tau_i \), the tasks in the system can be divided into four classifications: (1) the executing task, which has been released but it does not finish yet now. In a real-time system,
there is only one executing task. (2) the active tasks, which have been released but they are preempted by other tasks now. (3) waiting tasks, which are waiting to be released. (4) sleeping tasks, which have finished the work in current period and are waiting the new period.

Additionally, the task model in this paper has the following hypotheses: (1) The tasks are independent, that is to say there is no conflictive resource except CPU time and no dependent relationship between any two tasks. (2) None of the tasks could be suspended by itself. (3) The switch time between tasks is thought to be zero or very small so that it could be ignored.

3 Priority Assignment

3.1 Immediate Value

In a real-time system, a system function is implemented by executing a task, thus the importance of the task is embodied by the significance of the function. In this paper, we differ the importance of the tasks by labeling them with different values. Therefore, the value (denoted by \( V \)) could be thought as an inherent property of a task, and a task could bring as much value as \( V \) to the system as it is successfully submitted. However, the value that a task brings to the system isn’t generated just at the instant that the task commits, but is accumulated at a certain rate as the task executes. In order to detailedly expatiate the dynamic process that a task generates its value, we give the following definition:

**Definition 1. (Immediate Value)** The accumulative value of a task which has executed \( t \) time-units is named the immediate value of it, and denoted by \( IV(t) \).

If use a curve \( IV(t) = k \times t^p (p \geq 1) \) to express the process that a task accumulate its value, then there is \( V = k \times C^p \Rightarrow k = \frac{V}{C^p} \) when the task commits. Thus, the immediate value of a task can be gotten by the following formula:

\[
IV(t) = \frac{V \times t^p}{C^p} \tag{1}
\]

Obviously, the parameter \( p \) reflects the rate that a task produces its immediate value. If \( p = 1 \), the immediate value is symmetrically produced, but, if \( p > 1 \), it is produced at an accelerated rate.

However, the immediate value of a task is just thought to a possible gain to the real-time system. Only if the task successfully commits, the system could gain the value that equals the value of a task, or else the system discards the produced immediate value.

3.2 Remainder Value Density

**Definition 2. (Average Value Density)** The ratio of the expectant value and the computation time of a task is named its average value density, and denoted by \( AVD \).

The average value density, \( AVD = \frac{V}{C^p} \), is dependent on the inherent attributes of the task, and it reflects the contribution of the task to the system per time-unit. However, as the task produces its immediate value at an accelerative ratio, the average value density can not describe the varying process of producing the immediate value. Thus, the definition of remainder value density is given.

**Definition 3. (Remainder Value Density)** For a task which has executed \( t \) time-units, its average value produced per time-unit in the remainder computation time is named its remainder value density, and denoted by \( RVD(t) \).

Based on the definition 3, the remainder value density of a task can be calculated by the following formula:

\[
RVD(t) = \frac{V - IV(t)}{C - t} = \frac{(C^p - V) \times V}{C^p(C - t)} \tag{2}
\]

where \( V \) is the expectant value of the task \( IV(t) \) is the immediate value, \( C \) is the computation time.

Because \( RVD(t) = \frac{(C^p - t^p) \times V}{C^p(C - t)} = 1 - \left(\frac{t}{C}\right)^p \times \frac{V}{C - t} \), obviously, \( RVD(t) = AVD \) as \( p = 1 \).

**Theorem 1.** If \( t(0 \leq t < C) \) is fixed, the remainder value density, \( RVD(t) \), of a task increases as \( p(\geq 1) \) augments.

**Proof.** Let \( f(p) = \frac{C^p - t^p}{C^p} \). If we can prove that the function \( f(p) \) is increasing on \( p \), then the theorem will proved too.

Because \( f'(p) = \frac{t^p(\ln C - \ln t)}{C^p} > 0 \), so \( f(p) \) is an increase function on \( p \). Obviously, the theorem is proved. \( \Box \)

**Theorem 2.** If \( p(p > 1) \) is fixed, the remainder value density, \( RVD(t) \), of a task, increases with \( t(0 \leq t < C) \) increasing.

**Proof.** Let \( f(t) = \frac{C^p - t^p}{C^p} \). If we can prove that the function \( f(t) \) is increasing on \( t \), then the theorem will be proved too.

Based on \( f(t) \), we can get \( f'(t) = \frac{(p - 1)t^{p-1} + C^p - Cpt^{p-1}}{(C-t)^2} \).

If let \( g(t) = (p-1)t^p + C^p - Cpt^{p-1} \), then we have \( g'(t) = p(p-1)t^{p-2}(t-C) \).

Because \( p \geq 1 \) and \( 0 \leq t < C \), so \( g'(t) < 0 \) and \( g(t) \) is decreasing on \( t \). Thus there is \( g(t) > g(C) \Rightarrow g(t) > 0 \).

So \( f'(t) = \frac{(p-1)t^p + C^p - Cpt^{p-1}}{(C-t)^2} = \frac{g(t)}{(C-t)^2} > 0 \), that is, the \( f(t) \) is an increase function and the theorem is proved. \( \Box \)

Because only if a task is committed, the real-time system could gain the value produced by the task, or else, the system discards it. So if the task aborts, the system resources, such as the memory and the CPU time, that the task has spent are wasted without generating any value for the system. In order to avoid it, as a task begins, it should be protected from being aborted as possible as we can.
If the tasks’ priorities are differed based on their remainder
value densities, then the priority of the executing task
keeps increasing as its execution time \( t \) increases. However,
for an active or waiting task, its priority keeps changeless
for its execution time is unchanged. Therefore, the possi-
bility that the executing task is preempted and aborted is
greatly cut down, but the probability that the executing task
could be successfully finished is obviously improved.

If let \( r(t) = \frac{RVD(t)}{RV D(0)} \), here \( RV D(0) \) and \( RV D(t) \) are respec-
tively denoted the remainder value densities of a task as it
begins and it has executed \( t(0 < t < C) \) time-units. Obvi-
ously, there is \( RV D(0) = \frac{V}{C} = AVD. \) Based on the formula 2,
there is \( r(t) = \frac{(C^\prime - t)\nu}{C^\prime t(C - t)} \times \frac{C^\prime}{C} = \frac{C^\prime - t}{C - t} \times \frac{(C - t)^{p-1}}{C - t}. \) Obviously,
\( r(t) \equiv 1 \) if \( p = 1. \) Based on theorem 1, there is \( r(t) > 1 \) if
\( p > 1 \) and \( t > 0. \) And based on theorem 2, if \( q \) is fixed, the
remainder value density, \( RV D(t) \), reaches its maximum just
before the task commits according to the following formula.

\[
\max r(t) = \lim_{t\to C} r(t) = \lim_{t\to C} \frac{RV D(t)}{RV D(0)} = \frac{1}{C^\prime} \sum_{i=0}^{k} \frac{1}{(C - t)^{i}} \times C^\prime \]

\[
= p \quad (3)
\]
That is, for any a task, the value range of its remainder value
density is \( [AVD, p \times AVD]. \)

As analyzed above, the remainder value density of a
waiting task equals its average value density \( AVD \) for its
execution time \( t = 0 \), and which is independent of the value
of \( p. \) However, for an active or executive task, its remain-
der value density will increase as \( p \) augments as its execution
time \( t \) is fixed. Therefore, if the priorities of the tasks are
differed according to their remainder value densities, the
priorities of the executing task and the active tasks could be
improved by choosing a bigger value for \( p \), which could
help to make sure them could accomplished their jobs.

### 3.3 Urgency

If the priorities of the tasks are assigned according to
their value density, the task with higher value density could
win the CPU when scheduling the tasks. However a urgent
task but with low value density might miss its deadline for it
could win the CPU in time. In order to improve the holistic
performance of the real-time system, we should also con-
sider the effect of the urgency of a task on its priority. Next,
the task’s urgency will be defined and its characters will be
explored.

**Definition 4. (execution intensity).** For any task, suppose
it has executed \( t \) time-units at time \( \tau \), the proportion of its
required execution time to its available time is named its
execution intensity at that time, and denoted by \( \phi(t). \)

According to the definition, there is \( \phi(t) = \frac{C_{i} - t}{\tau_{i} - t} \) and
\( \phi(t) \leq 1. \) For any task, the higher its execution intensity
is, the more time it should spend to finish its job, and
then the earlier it should start in order to avoid being de-
layed (preempted by other tasks) or aborted. We define the
degree that a task should be executed early as the urgency
(denoted by \( \delta(t) \)) of the task, and calculate it according to the
following formula.

\[
\delta(t) = q \phi(t) = q \frac{C_{i} - t}{\tau_{i} - t}, \quad (4)
\]
here, \( t \) is the execution time, \( \tau \) is the current time , and
\( q(q \geq 1) \) is a parameter that could adjust the effect of
the urgency on the priority of the task. Obviously, the value
range of \( \delta(t) \) is \( [q \frac{C_{i}}{\tau_{i}}, q]. \) Specially, if \( q = 1, \) it means that
we do not consider the urgency of a task when assigning its
priority.

**Theorem 3.** If \( q > 1, \) the urgency \( \delta(t) \) of a task increases
as it waits more time.

**Proof.** Suppose at time \( \tau, \) task \( T_{1} \) has accumulatively
waited \( x \) unit-times (excluding its execution time), so \( d_{i} -
\tau_{i} = D_{i} - x - t(0 \leq x < D_{i} - t, 0 \leq t < C_{i}). \) Then the urgency
of \( T_{1} \) can be rewritten to be \( \delta(t) = q \frac{C_{i} - t}{\tau_{i} - t} (q > 1). \)
Let \( f(x) = \frac{C_{i} - t}{D_{i} - x - t} \), then \( \delta(t) = \frac{q}{q} f(x). \) We could
prove that \( f(x) \) is an increasing on \( x, \) then it is obvious that
\( \delta(t) \) increases on \( x. \)

Because \( f(x) = \frac{C_{i} - t}{(D_{i} - t - x)} > 0, \) so the function \( f(x) \)
increases on \( x. \) Thus the theorem is proved.

In a real-time system, the tasks are different on the ur-
gencies because of their different time attributes. Thus
four principles are available to calculate the urgencies of the
tasks: (1) Only the task which could catch its deadline
should be consider when computing the urgency. (2) The
urgency of the executing task keeps changeless and equals
that when it begins to execute in the recent time. (3) For an
active or a waiting task, it urgency increases as time passes.
(4) Omitting the urgencies of the sleeping tasks for they
have not been released yet.

### 3.4 Dynamic Priority Assignment

By now, we have introduced the remainder value den-
sity and the urgency of a task and the effects of them on
the priority of the task. As has analyzed above, if the tasks
are scheduled according to the priorities of their remainder
value density, the task with higher value contribution will be
executed with higher priority, however, the task with lower
value contribute but higher urgency might miss its deadline
for it is hard to win the CPU. On the contrary, if scheduling
the tasks based on their urgencies, the urgent task might be
executed first, but the loose task with high value contribution might miss its deadline for it has no chance to execute.

However, in order to improve the whole performance of the real-time system, we need consider the effects both the value densities and the urgencies of the tasks on their priorities when scheduling the tasks. Thus, a dynamic priority assignment strategy based on the value density and urgency are proposed. Suppose $T_i$ is a non-sleeping task, and it has executed $t$ unit-times at time $\tau_i$, then the priority, $Pri(T_i)$, of which could be computed by the following formula:

$$Pri(T_i) = RVD_i(t) \times \delta_i(t) = \frac{(C_i^p - t^p)}{C_i^q (C_i - t)} \times q \times \frac{C_i - \tau_i}{C_i - \tau_{\max}}$$

(5)

Based on formula 5, when $T_i$ is released, its remainder value density and urgency are least, so its priority is the minimum (denoted by $\min Pri(T_i)$) too. Because there are $t = 0$ and $D_i = d_i - \tau_i$, when $T_i$ is released, so $\min Pri(T_i) = \min(RVD_i(t)) \times \min(\delta_i(t)) = V_i \times \frac{C_i}{C_i^q} \times q \times \frac{C_i - \tau_i}{C_i - \tau_{\max}}$.

If $C_i \leq D_i$, then $C_i \times t \rightarrow 0$, an then $\min Pri(T_i)$ is very close to $V_i$. Obviously, the minimal priority of a task is just determined by its basic attributions, and we name it as the basic priority of the task. After $T_i$ is released, with the waiting time or execution time increasing as time goes by, the remainder value density or urgency of $T_i$ is augmented, and the priority of which keeps increasing too. Suppose task $T_i$ is accomplished just before its deadline, its remainder value density and the urgency get to their maximum at the same time. Then the priority of $T_i$ reaches its maximum too and is denoted by $\max Pri(T_i)$. At that time, there are $t \rightarrow C_i$ and $(d_i - \tau_i) \rightarrow 0$, so $\max Pri(T_i) = \max(RVD_i(t)) \times \max(\delta_i(t)) = pq \times \frac{V_i}{C_i}$.

So during the execution period of $T_i$, its priority is ranged from $\frac{V_i}{C_i}$ to $pq \times \frac{V_i}{C_i}$. Except $C_i$ and $V_i$, the priority of a task is also affected by the value of $p$ and $q$ after the task is released. Thus, we could adjust the value of the priority of a task by choosing different value for $p$ and $q$.

4 Preemptive Scheduling Strategy

4.1 Thrashing and Avoidance

When scheduling the tasks in a real-time system based on their dynamic priorities, due to the priorities of the tasks increases at varying rate, it is possible that two or more tasks might be frequently preempted each other. Such phenomena is called system thrashing [5]. Usually, some system resources must be spent on context switching as a preemption is happened, and a system thrashing means there are many frequent preemptions between the tasks during very short time. So a lot of system resources might be wasted when a thrashing happens. In order to improve the system performance, thrashing must be avoided.

Suppose $T_1$ and $T_2$ are two tasks in the real-time system, $T_1$ is the executing task, $T_2$ is a waiting or active task, and their priorities are respectively denoted by $Pri(T_1)$ and $Pri(T_2)$ at time $\tau_i$. If give a parameter $\beta (\beta \geq 1)$, and only if the condition holds,

$$Pri(T_2) \geq \beta \times Pri(T_1)$$

(6)

$T_2$ is permitted to preempt $T_1$, then based on the following theorem 4, we always could find a value for $\beta$ to make sure that the thrashing could be avoided.

Theorem 4. Given a real-time task set, it always could find a value for $\beta$ to avoid the system thrashing.

Proof. Suppose $T_1$ and $T_2$ are two tasks in a real-time system, the average value density of $T_1$ is maximal in the system and denoted by $AVD_1$, that of $T_2$ is the minimal and denoted by $AVD_2$. At time $\tau_i$, $T_1$ is the executing task, $T_2$ is a waiting or active task, and their priorities at that time are respectively denoted by $Pri(T_1)$ and $Pri(T_2)$. If $Pri(T_1)$ and $Pri(T_2)$ satisfy the condition in the formula 6, then $T_2$ is permitted to preempt $T_1$. In order to avoid the possible system thrashing, it must make sure that $T_2$ could not be preempted by $T_1$ after $x$ time-units, where $x$ is bigger than 0 and less than rest execution time of $T_1$ and $T_2$.

Suppose the priorities of $T_1$ and $T_2$ are respectively $Pri'(T_1)$ and $Pri'(T_2)$ after $x$ time-units, in order to avoid the system thrashing, the following condition must hold.

$$\frac{Pri'(T_1)}{Pri'(T_2)} \Rightarrow \beta \geq \max \left( \frac{Pri'(T_1)}{Pri'(T_2)} \right)$$

(7)

Suppose at time $\tau_i$, the remainder value density of $T_1$ reaches its maximum $p \times AVD$. After that, it is preempted by $T_2$, and in the following $x$ time-units, its urgency keeps increasing till to be the maximum $q$. But for $T_2$, it has the minimal urgency $q \frac{C_i}{D_i}$ and the minimal remainder value density $AVD$ at time $\tau_i$. After that, $T_2$ changes to be the executing task and its urgency keeps changeless. Although the remainder value density of which will increase in the next $x$ time-units, we can think that $x$ is so small that the amplitude of increase is rather small and could be omitted. Thus, there is $\beta \geq \max \left( \frac{Pri'(T_1)}{Pri'(T_2)} \right) \Rightarrow \beta \geq \frac{AVD_1 \times p \times x}{AVD \times q} \geq \frac{AVD_1 \times p \times x}{AVD_2 \times q}$. Because there is $\beta \geq 1$, so the value of $\beta$ should satisfy the following condition.

$$\beta \geq \max \left( 1, \frac{AVD_1 \times p \times x}{AVD_2 \times q} \right)$$

(8)

Based on the above analysis, the thrashing could be avoided if the $\beta$ is chosen according to the formula 8. □
4.2 Scheduling Strategy Based on Dynamic Priority

When scheduling the tasks in a real-time system, in order to improve the performance of the system, the task which has high contribution to the system or urgent should be scheduled to be executed first. Thus this paper proposes a preemptive scheduling strategy based on dynamic priority.

Suppose $T_1$ is the executing task, $T_2$ is the task with the maximal priority among of the active and waiting tasks. At time $\tau_i$, $T_1$ has executed $t_1(0 \leq t_1 < C_1)$ time-units and there is $d_1 - \tau_i \geq C_1 - t_1$, that is to say it could satisfy its deadline. $T_2$ has executed $t_2(0 \leq t_2 < C_2)$ time-units and $d_2 - \tau_i \geq C_2 - t_2$, that is also to say it could satisfy its deadline too. If the priorities of $T_1$ and $T_2$ are respectively denoted by $Pri(T_1)$ and $Pri(T_2)$, then the preemptive scheduling strategies could be depicted as follows.

1. If $Pri(T_2) < \beta \times Pri(T_1)$, then $T_2$ should not preempt $T_1$ in order to avoid thrashing. So $T_1$ continues to execute and $T_2$ keeps its state.

2. If $Pri(T_2) \geq \beta \times Pri(T_1)$, then the condition that $T_2$ could preempt $T_1$ holds. Two different strategies are available according to the deadline of $T_2$.

- If both $d_2 - \tau_i - (C_1 - t_1) \geq C_2 - t_2$ and $d_1 - \tau_i - (C_2 - t_2) \geq C_1 - t_1$ hold, $T_2$ could catch its deadline even if it begins to execute till after $T_1$ is accomplished, and $T_1$ could also catch its deadline even though it is delayed till $T_2$ finishes its jobs. Under this situation, the strategy I in section 4.2.1 is suitable.

- If only the condition $d_2 - \tau_i - (C_1 - t_1) < C_2 - t_2$ holds, then $T_2$ will miss its deadline if it does not preempt $T_1$, so $T_2$ must preempt $T_1$. Under this situation, two strategies are alternative according to whether $T_1$ is abort or not after it is preempted: (a) If $d_1 - \tau_i - (C_2 - t_2) \geq C_1 - t_1$ holds, then $T_1$ could catch its deadline even though it is preempted by $T_2$. To deal with this situation, the strategy II in section 4.2.2 is available. (b) If $d_1 - \tau_i - (C_2 - t_2) < C_1 - t_1$ holds, $T_1$ will miss its deadline as it is preempted by $T_2$, and the strategy III in section 4.2.3 is available.

4.2.1 Strategy I

If the condition that $T_2$ could preempt $T_1$ is held, and both $T_1$ and $T_2$ could catch their deadlines whether $T_2$ preempts $T_1$ or not, then the strategy I is used to schedule $T_1$ and $T_2$. Here, two different strategies are alternative.

1. Optimistic strategy. The optimistic strategy believes that $T_2$ could catch its deadline even though it does not preempt $T_1$, because it is low probability that a task with higher priority will appear during $T_2$ waits $T_1$. Thus, the optimistic strategy decides that $T_1$ continues to execute and $T_2$ keeps waiting. Obviously, the optimistic strategy could avoid unnecessary context switch, and which could improve the performance of the system.

2. Pessimistic strategy. The pessimistic strategy believes that $T_2$ will miss the chance to win the CPU if it does not preempt $T_1$, because it is high probability that the tasks with higher priorities might be released during $T_2$ waits $T_1$. Thus $T_2$ should preempt $T_1$ in order to obtain higher system performance. The concrete method of this strategy is same as the strategy II introduced in section 4.2.2.

4.2.2 Strategy II

If $T_2$ must preempt $T_1$, and $T_1$ still could catch its deadline even though it continues to execute till $T_2$ finishes, then the strategy II is available. The method of the strategy II is very easy, $T_2$ preempts $T_1$ and $T_1$ is delayed till $T_2$ finishes.

4.2.3 Strategy III

If $T_2$ must preempt $T_1$ and it will cause $T_1$ to be aborted, then the strategy III is available.

Because $T_1$ has generated some value after it executes $t_1$ time-units and spends a quantity of system resources, if $T_1$ is preempted and aborted, the generated value by which will be discarded by the system. And the resources have been spent by $T_1$ are believed to be wasted for the system. Thus, if $T_2$ preempts $T_1$, it must compensate the system for the lose due to $T_1$ is aborted. Then, at time $\tau_i$, if $T_2$ must preempt $T_1$, it should satisfy the following condition.

$$\frac{C_2^* - t_2^*}{C_2^* - t_2} \times (V_2 - \frac{V_1 \times t_1^*}{C_1^*}) \times q \frac{C_2 - t_2}{C_2 - t_2} \geq \beta \times \frac{C_1^* - t_1^*}{C_1^* - t_1^*} \times V_1 \times q \frac{C_1 - t_1}{C_1 - t_1},$$

(9)

here, $\frac{V_1 \times t_1^*}{C_1^*}$ denotes the immediate value of $T_1$ at time $\tau_i$.

5 Simulation

All the experiments were conducted on a pc with 2.0 GHZ CPU, 1 GB memory and running WinXP. All codes were complied using Visual C++. All tasks used in the experiments were randomly selected. In the simulations, we compare our algorithm with EDF [7], HVF [4] and HVDF [1] method by six experiments whose load ranging from 0.5 to 3.0. In each experiment, three task sets are experimented. Three criteria, Total Value(TV), Preemptive Number(PN) and Miss Deadline Ratio(MDR), are used to evaluate the performance of the methods.

As shown in figure 1, when the load of the real-time system is less than 1.0, the TV and MDR of the system using our method are worse than those using EDF algorithm. The main reason is that all the tasks could accomplished if they
Figure 1. The performance comparison of the four methods

are properly scheduled according to their deadlines, and EDF is a competent algorithm. And in the value(density) based algorithms, the task with small value(density) but low urgency might miss its deadline. However, when the load of the system is bigger than 1.0, only a part of tasks could catch their deadline due to the capability of the system is limited. Then which task is executed first will greatly effect the performance of the system. Compared with the value(density) based algorithms, the EDF method could improve the MDR and PN of the system, but the value(density) based algorithms could gain bigger value than the EDF method. However, the effects of the value density and the urgency of a task on its priority are considered when scheduling the tasks using our method. So as shown in figure 1-(a) and 1-(b), the TV and MDR of our method is prior to the other three methods when the load is bigger than 1.0. Furthermore, because our method improves the difficulty of preemption by setting the preemptive threshold, so the number of preemption is greatly cut down. As shown in figure 1-(c), the PN of our method is the better than that of the other methods no matter what the system load is.

6 Conclusion

The paper presents a preemptive scheduling algorithm to schedule the tasks in a real-time system. According to the algorithm, the weights of the value density and urgency of a task impact on its priority are adjusted by two parameters p and q. Moreover, the parameter β is properly chosen to avoid system thrashing. The experiment results of the simulations show that the algorithm is prior to the analogous algorithms.

7 Acknowledgement

The work of this paper was supported by the National Natural Science Foundation of China under Grant No.60763002 and the Natural Science Foundation of Jiangxi Province of China under Grant No.2008GZS0021.

References