Prioritizing Soft Real-Time Tasks to Improve End-to-End Response Time in Distributed Control Systems

Namyun Kim, Taewoong Kim, Naehyuck Chang and Heonshik Shin

Department of Computer Engineering, Seoul National University
Seoul 151-742, Korea
e-mail: {nykim,twkim,naehyuck,shinhs}@comp.snu.ac.kr

Abstract. In distributed control systems, real-time tasks may be classified into local tasks and global tasks. Local tasks are usually bound to a specific node and responsible for hard real-time functions to control I/O devices whereas global tasks perform soft real-time functions such as data logging, status reporting, and statistical analysis. Such a global task is divided into several subtasks each of which is allocated to a node and has precedence relation with other subtasks. To schedule local tasks and subtasks in a node, we must determine their priorities in the fixed-priority system. In this paper, we present an algorithm to assign priorities to tasks to guarantee the deadlines of local tasks and improve the end-to-end response time of a global task which amounts to the sum of response time of its subtasks. The proposed algorithm has no run-time overhead such as slack calculation and its performance is comparable to that of slack stealing algorithm under various workloads.

1 Introduction

With the maturity of network and microprocessor technologies, large scale industrial systems have been developed in a distributed computing environment. The motor control system in factory automation, for example, deploys multiple nodes each of which controls a cluster of motors to ensure their proper operation, and communicates with the supervising node over LAN as shown in Fig. 1. Each controller consists of sensors, control program and actuators. The control program has multiple control loops which are periodically invoked and executed to update actuators in response to data sampled from sensors. Such a control loop has hard real-time constraints from reading sensors to updating actuators. Besides control loops, the data logging loop is used to gather data about the process operation. The information could be used for configuration control, performance logging, and diagnostic purposes to find out why a particular failure occurred. This information is transferred to a supervising node which periodically monitors the state of controllers, stores it into database. To update the status of controllers, supervisoring node sometimes sends control command to controllers. In such a model, tasks may be classified into *local* and *global* task[1]. A local task is executed on a node only. For example, a control loop in a controller

[1] They are also called end-to-end tasks

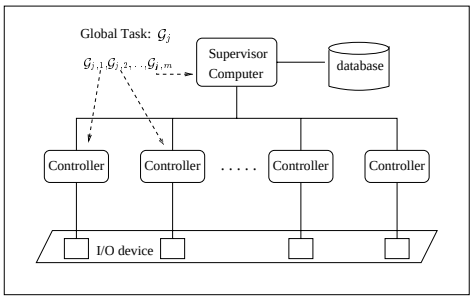

Fig. 1. The example of a distributed control system

corresponds to a local task. On the other hand, a global task is executed on multiple nodes. As a example, a global task is comprised of a data logging loop in a controller and associated monitoring task in the supervising node.

As timing constraints of a global task, application designers specify an end-to-end deadline. Since the unit of scheduling is the task, we must determine the priority or deadline of a subtask from the end-to-end deadline in order to schedule local tasks and subtasks in a node. To address this problem, several approaches derived the deadline of a subtask from end-to-end deadline[3, 7] since most of scheduling algorithms use the deadline as a measure of priority. Some of these approaches divide the overall slack of a global task and assign slack plus computation time to the deadline of a subtask[3]. The other approach decomposes end-to-end deadlines using a simple linear response time model[7]. However, these approaches have serious problem for hard real-time system; when tasks are scheduled by deadline driven algorithm, the improper deadline assignment of subtasks may cause local tasks to miss their deadlines. As a simple approach to schedule subtasks without missing the deadlines of local tasks, subtasks are assigned lower priorities than local tasks with hard real-time constraints. In this case, local tasks are not affected by subtasks. However, subtasks experience long response time.

In this paper, we present an off-line algorithm to assign a priority to each subtask in order to reduce the response time of a global task while meeting the deadlines of local tasks. Each node consists of a set of local tasks and a set of subtasks. We first sort each task set by deadline, merge two sets of tasks and generate the priorities of tasks. Our approach raises the priorities of subtasks using the slack of local tasks.

The rest of the paper is organized as follows. Section 2 describes the task model and Section 3 defines our problem. Then, Section 4 describes the algorithm to assign the priorities to tasks and Section 5 presents the schedulability analysis for task sets with offsets. Section 6 describes the simulation results on the proposed algorithm. Finally, in Section 7 we conclude the paper with summary and future research direction.

2 Task Model

We assume that local task has hard real-time constraints whereas a global task has soft real-time constraints. Furthermore, we assume all tasks are periodic and independent. When the supervising node sends command to a controller, its task may be modeled an aperiodic task. In that case, we assume the aperiodic task can be processed as a background or by a local periodic server [9].

Each task is characterized by offset, worst case computation time, deadline and period. Table 1 presents notation used in this paper. The attribute of local tasks is specified by application designer and thus is known in advance. We assume local tasks are sorted by their deadlines and schedulable under a given priority assignment. For a global task, application designer specifies end-to-end timing constraints. Thus, we have to assign task attributes to each subtask from end-to-end timing constraints.

$\mathcal{G}_j$ is a set of several related subtasks $\{\mathcal{G}_{j,1}, \mathcal{G}_{j,2}, \ldots, \mathcal{G}_{j,m}\}$ each of which is allocated to a different node. Each subtask has precedence relation with other subtasks. To guarantee the precedence relation, we define a recurrence relation as follows:

$$o_{j,1} = o_j, \quad o_{j,k} = o_{j,k-1} + r_{j,k-1} \; for \; k = 2, \ldots, m. \tag{1}$$

$r_{j,k-1}$ is obtained from off-line analysis after the priority of $\mathcal{G}_{j,k-1}$ is determined. Note that $o_{j,k}$ doesn't depend on $d_{j,k-1}$ but $r_{j,k-1}$. This has two meanings. First, if $o_{j,k}$ depends on $d_{j,k-1}$, $\mathcal{G}_{j,k-1}$ must be completed by $d_{j,k-1}$. Otherwise, $\mathcal{G}_{j,k}$ may be running before the completion of $\mathcal{G}_{j,k-1}$ and thus may result in reading incorrect data. However, missing the deadline of each subtask in a node may be acceptable since the global task performs the soft real-time function. Second, if $\mathcal{G}_{j,k-1}$ completes late, $o_{j,k}$ has large value and small slack. Thus, $\mathcal{G}_{j,k}$ can be assigned a higher priority compared with other subtasks in different global task. This results in the reduction of variance of the worst case response times of global tasks. It will be clear in the rest of the paper.

Table 1. Summary of Notation

Notation	Description
$\mathcal{L}_i$, $\mathcal{G}_j$	local task i, global task j
$\mathcal{G}_{j,k}$	the kth subtask of $\mathcal{G}_j$
o_j, $o_{j,k}$	offset of $\mathcal{G}_j$, $\mathcal{G}_{j,k}$
d_j, $d_{j,k}$	relative deadline of $\mathcal{G}_j$, $\mathcal{G}_{j,k}$
r_j, $r_{j,k}$	the worst case response time of $\mathcal{G}_j$, $\mathcal{G}_{j,k}$
$c_{j,k}$, $T_{j,k}$	maximum computation time and period of $\mathcal{G}_{j,k}$
$s_{j,k}$, $e_{j,k}$	slack, earliness of $\mathcal{G}_{j,k}$
N, M	the number of local tasks and subtasks in a node, respectively
Λ, Γ	a set of local tasks and subtasks in a node, respectively

All subtasks which belong to a global task have the same period. The deadline $d_{j,k}$ is derived from end-to-end deadline d_j. It is used to sort subtasks in a node.

The method to derive $d_{j,k}$ is described in Section 4.2. For a notation pertaining to a subtask, we define the *earliness* $e_{j,k}$ as $d_{j,k} - r_{j,k}$, to determine how early the subtask completes.

The message is treated analogously with a task. Its parameters include offset, the maximum message transmission time, deadline and period. In this paper, we will treat the network like a processor.

3 Problem Statement

The objective of this paper is to reduce the end-to-end response time of global tasks while meeting the deadlines of local tasks. The response time of a global task is the sum of the response time of its subtasks. To reduce the response time of global tasks, we assign the highest possible priority to each subtask using the slack of local tasks.

Given a set of local tasks and a set of subtasks in a node, the optimal priority assignment is to maximize the total earliness of subtasks. Earliness is a measure to show how early the subtask completes on the basis of its deadline. Under assumption that local tasks are sorted by its deadline, the optimal priority assignment can be obtained by inspecting $\frac{(N+M)!}{N!}$ priority sequences. However, it is intractable to obtain the optimal priority assignment due to large search space. To reduce the search space, we sort a set of subtasks according to their deadline. Then we merge a set of local and subtasks to produce a priority sequence such that local tasks are schedulable and the highest possible priority for each subtask is obtained.

The reason to sort a set of subtasks is that we assign higher priority to a subtask with a shorter deadline. This, in effect, reduces the response time of the subtask with shorter deadline, with potential increase in the response time of the other subtasks. Since $e_{j,k}$ is defined as $d_{j,k} - r_{j,k}$, we can increase the minimum earliness by doing so.

4 Proposed Scheme

4.1 Ordering of Nodes

In order to determine the priority of a subtask $\mathcal{G}_{j,k}$, it is necessary to know its timing constraints, such as $o_{j,k}$, $c_{j,k}$, $d_{j,k}$, and $T_{j,k}$. However, $o_{j,k}$ is not known before $r_{j,k-1}$ is determined. To determine $r_{j,k-1}$, we have to first assign a priority to $\mathcal{G}_{j,k-1}$. Thus, we should decide the order of nodes to perform the priority assignment procedure.

In case that a node communicates with other nodes, we can analyze the communication pattern of the nodes and then build the precedence graph off-line. In this paper, we assume the precedence graph is a DAG(directed acyclic graph) for which we can find a consistent labeling of nodes[8]. A consistent labeling is that if there is an edge from node v to node w, the inequality $v < w$ holds. Once the priorities of subtasks have been determined in nodes with smaller label than w, the offset of all subtasks in a node w is known. Thus, we can determine priorities of subtasks in a node w.

4.2 Determining Deadline of Subtask

The deadline can be determined by several approaches presented in [3, 7]. A simple method is to define the deadline of a subtask as the sum of its allocated slack and computation time. The slack of a subtask, $s_{j,k}$ can be obtained by simply dividing the overall slack equally among all subtasks of a global task. Let the set of subtasks which have not been assigned a priority be $\{ \mathcal{G}_{j,k}, \mathcal{G}_{j,k+1}, \ldots, \mathcal{G}_{j,m} \}$. Then, $s_{j,k}$ and $d_{j,k}$ can be calculated as follows:

$$s_{j,k} = \frac{(d_j - o_{j,k} - \sum_{l=k}^{m} c_{j,l})}{(m-k+1)}, \quad d_{j,k} = s_{j,k} + c_{j,k} \tag{2}$$

The late completion of $\mathcal{G}_{j,k-1}$ increases $r_{j,k-1}$ and thus $\mathcal{G}_{j,k}$ has large $o_{j,k}$. This causes $s_{j,k}$ to be small and $\mathcal{G}_{j,k}$ to have a shorter deadline. Thus, $\mathcal{G}_{j,k}$ may be assigned a higher priority.

4.3 Priority Assignment Algorithm

In order of node determined in Section 4.1, we execute the priority assignment algorithm off-line. Suppose there are a set of N local tasks, $\Lambda = \{\mathcal{L}_i\}$ and a set of M subtasks, $\Gamma = \{\mathcal{G}_{j,k}\}$ in a node. Λ is sorted by increasing order of deadline of local task and is assumed to be schedulable in this order. Γ is sorted by increasing order of deadline of subtask. The priority assignment algorithm merges each set of tasks into a priority sequence. The number of possible priority sequences is $\frac{(N+M)!}{N! \times M!}$. Among these, we will find a priority sequence such that a set of local tasks are schedulable and the highest possible priority for each subtask is obtained.

A simple method to obtain such a priority sequence is to add each subtask to the ordered set of local tasks and perform the schedulability test. Let Φ_i be the ith local task in Λ. Each subtask in Γ is removed in increasing order of its deadline(i.e., subtask with the shortest deadline is chosen first) and is placed in front of Φ_N in Λ. Then the schedulability test is performed for Φ_N assuming that tasks preceding Φ_N have higher priority. If the test succeeds, the chosen subtask is placed in front of Φ_{N-1}. Then, schedulability test is repeated for Φ_{N-1}. This continues until the schedulability test fails. If it does, the chosen subtask is replaced in the previous place. At this moment, Λ is extended by one and the position(i.e., priority) of the chosen subtask is determined. Note that the deadlines of all local tasks in Λ are met, since the added subtask cannot affect local tasks in former position than its position and the schedulability has been tested for local tasks following it. The above process is repeated for the next subtask in Γ until positions of all subtasks in Γ are determined. As a matter that demands special attention, the next subtask should be placed behind the previous subtasks since we assign the higher priority to subtask with shorter deadline. According to the position of tasks, we assign the priority to tasks.

The above scheme is simple, but its complexity is high since it finds an appropriate position for each subtask. The worst case occurs when any local task meets its deadline regardless of position(i.e., priority) of a subtask. In this case, the number of schedulability tests is N for each subtask. Therefore, the

total number of tests is $M \times N$. If we check the schedulability for a local task with all subtasks which may have higher priorities, the complexity can be reduced. The improved algorithm works as follows. First, all subtasks in Γ are placed in front of Φ_N in Λ. Then we check the schedulability for Φ_N assuming that tasks preceding Φ_N have higher priority. If Φ_N is schedulable, Φ_N is assigned the lowest priority $N + M$. Then, all subtasks in Γ are promoted one place and the schedulability test is performed for Φ_{N-1}. If Φ_N is not schedulable, remove a subtask with the largest deadline from Γ one at a time until Φ_N is schedulable. Then we assign removed subtasks the priorities from $N + M$ to $N + M - l_n + 1$, where l_n is the number of removed subtasks. Φ_N is assigned the priority $N + M - l_n$. The above process is repeated for Φ_{N-1} until Γ or Λ is empty. Finally, each task in non-empty set is assigned a unique priority.

In this approach, when the schedulability test for Φ_i fails, a subtask is removed and its priority is assigned. To the contrary, when the test succeeds, the subtasks in Γ are promoted one step and the priority of Φ_i is assigned. Thus, the number of schedulability tests for Φ_i is the number of removed subtasks and plus one. Finally, the total number of tests is

$$(1 + l_1) + (1 + l_2) + \ldots + (1 + l_N) = N + \sum_{i=1}^{N} l_i \tag{3}$$

As $\sum_{i=1}^{N} l_i$ is at most M, its upper bound is reduced to $M + N$. Fig. 2 describes the procedure for priority assignment.

So far, we have described priority assignment method for tasks. For messages to be communicated to other nodes, priority assignment is trivial since only subtasks generate the message. Thus, the priority of a message is determined by deadline.

Algorithm

Variable:
π : the priority level Δ_i : a set of higher priority tasks than $\mathcal{L}_i$

$\pi = N+M$;
while(Γ is not empty)
  remove $\mathcal{L}_i$ with the largest deadline from Λ;
  $\Delta_i = \Gamma \cup \Lambda$; check schedulability of $\mathcal{L}_i$ with Δ_i;
  while($\mathcal{L}_i$ is not schedulable)
    remove $\mathcal{G}_{j,k}$ with the largest deadline from Γ;
    assign π to $\mathcal{G}_{j,k}$; decrease π by one ;
    if(Γ is empty) assign priority to remaining tasks in Λ; return;
    $\Delta_i = \Gamma \cup \Lambda$; check schedulability of $\mathcal{L}_i$ with Δ_i;
  endwhile
  assign π to $\mathcal{L}_i$; decrease π by one;
  if(Λ is empty) assign priority to remaining tasks in Γ; return;
endwhile

Fig. 2. Priority assignment of tasks in each node

5 Schedulability Analysis

The simple analysis of preemptive task sets with offsets can be carried out by ignoring the offsets [2]. In this case, a critical instant for any task occurs at time 0. Thus, the worst case response time obtained at time 0 is given by:

$$r_i(n) = c_i + \sum_{\forall j \in (hp(i))} (\lceil \frac{r_i(n-1)}{T_j} \rceil \times c_j) \tag{4}$$

where $hp(i)$ is the set of all tasks of higher priority than a task i. The iteration begins with $r_i(0) = 0$ and continues until the equation converges to a finite value. If $r_i(n) \leq d_i$ for all tasks, we can state that a task set is schedulable. Although this analysis is simple, it can produce pessimistic results for tasks with offsets if tasks are not simultaneously released.

To accurately determine the schedulability of a task we need to check to see if every individual release of a task is schedulable. Leung [6] shows that a task set is feasible if every release of all tasks is schedulable over the interval [$max(o_i), 2 \times lcm(T_i)$) for dynamic priority scheduling schemes. For static priority schemes, Audsley [1] improved the previous result by limiting the interval to [$max(o_i), lcm(T_i) + max(o_i)$). The schedulability test in [1] requires the construction of a schedule for this interval. This approach creates a set of tuple (c_i, t) which consists of computation time and release time for every task and sorts them in non-decreasing release time, which may be inefficient. In this paper, we provides a method for examining schedulability without constructing a schedule.

For a release of task i at t, the computation demand of higher priority tasks and task i is divided into three parts: (1) the computation time of task i, (2) the remaining computation time of higher priority tasks which have arrived before t and have not completed their computation at t, Q_i^t, (3) the computation time of higher priority tasks which are released after t. Now, we can express the worst case response time of task i at a release time t as follows:

$$r_i(n) = c_i + Q_i^t + \sum_{\forall j \in hp(i)} [(\lceil \frac{t + r_i(n-1) - o_j}{T_j} \rceil - \lceil \frac{t - o_j}{T_j} \rceil) \times c_j] \tag{5}$$

The term $\lceil \frac{t-o_j}{T_j} \rceil$ and $\lceil \frac{t+r_i(n-1)-o_j}{T_j} \rceil$ are the number of releases of task j for the interval $[o_j, t)$ and $[o_j, t+r_i(n-1))$, respectively. Thus, the number of releases for the interval $[t, t + r_i(n-1))$ is $\lceil \frac{t+r_i(n-1)-o_j}{T_j} \rceil - \lceil \frac{t-o_j}{T_j} \rceil$. When $r_i(n) = r_i(n-1)$, task i is schedulable at this release if $r_i(n) \leq d_i$.

Q_i^t can be determined by examining the level $i-1$ busy periods for tasks which are released at the interval $[f_i$, t) where f_i is denoted as the completion time of task i at the previous release. Note that all higher priority tasks which are released before f_i have completed. Otherwise, the completion time of task i is larger than the value of f_i. The level $i-1$ busy period is a time interval within which the priority of the processor is higher than that of task i [4]. Let u and w be the start time and duration of level $i-1$ busy period, respectively.

The duration of the level $i-1$ busy period starting at u is calculated as follows:

$$w(n) = \sum_{\forall j \in hp(i)} [(\lceil \frac{\min(u + w(n-1), t-1) + 1 - o_j}{T_j} \rceil - \lceil \frac{u - o_j}{T_j} \rceil) \times c_j] \quad (6)$$

One reason for obtaining the minimum of $u + w(n-1)$ and $t-1$ is that we consider only tasks released before t. The number of releases of task j in the interval $[u, \min(u + w(n-1), t-1) + 1)$ is $\lceil \frac{\min(u+w(n-1),t-1)+1-o_j}{T_j} \rceil - \lceil \frac{u-o_j}{T_j} \rceil$. The term *+1* in the formula is due to the fact that when the iteration is in the initial state, i.e., $w(0)$ =0, we should consider tasks released at u. The other reason is that if two adjacent level $i-1$ busy periods, for example [a,b] and [b,c], exist, it is possible to merge two periods into [a,c]. If $w(n) = w(n-1)$, i.e., the equation has converged, the duration of busy period, w is equal to $w(n)$. There may be many level $i-1$ busy periods for the interval [f_i,t). To determine Q_i^t, we step through the level $i-1$ busy period for tasks released at the interval $[f_i, t)$ until $u < t$ and $u + w < t$. If $u \geq t$, all higher priority tasks released before t have completed and thus Q_i^t is 0. If $u + w \geq t$, we can obtain Q_i^t equal to $w - (t - u)$. The process to determine Q_i^t is summarized as below(initially, $u = f_i$):

1. The start time of level $i-1$ busy period is the minimum of the release times of all higher priority tasks from u.

$$u = \min_{\forall j \in hp(i)} (\lceil \frac{u - o_j}{T_j} \rceil \times T_j + o_j) \quad (7)$$

   If $u \geq t$, Q_i^t =0 and the process stops. Otherwise, go to step 2.
2. We calculate the duration of the level $i-1$ busy period starting at u using Eq. (6). When the equation converges, w is equal to $w(n)$. If $u + w < t$, $u = u + w$ and go to step 1. Otherwise $Q_i^t = w - t + u$ and the process stops.

6 Simulation Results

In this section, we evaluate the performance of the proposed algorithm by measuring the worst case response time of global tasks. The simulation is conducted for three scheduling algorithms: slack stealing algorithm[5], proposed algorithm and background algorithm. The slack stealing algorithm is an algorithm for servicing soft aperiodic tasks in a real-time system in which hard periodic tasks are scheduled using a fixed priority algorithm. Though the slack stealing algorithm requires a relatively large amount of calculation, it provides lower bound on the response time of aperiodic tasks. Since a periodic task can be represented as a set of aperiodic tasks, we choose the slack stealing algorithm as lower bound on the response time of soft real-time subtasks. In the background algorithm, all subtasks have lower priorities than any local task. It is chosen as upper bound on the response time of subtasks.

In this experiment, we consider a distributed system with six nodes. There are three global tasks in the system. We assume that each global task consists of six subtasks. Each subtask of a global task is executed at a different node. In

addition to subtasks, there are four local tasks in a node. Thus all algorithms have been executed for a task set consisting of four local tasks and three subtasks in a node. The assumption about the parameter of each task is as follows:

- The offset of all local tasks and global tasks is zero. The offset of a subtask is determined by the completion time of preceding subtask.
- We assume deadlines of local and global tasks are identical to their period.
- Local tasks at nodes 1, 3 and 5 have the periods of 40, 50, 60 and 80. Local tasks at the other nodes have the periods of 20, 40, 50 and 80.
- The periods of $\mathcal{G}_1$, $\mathcal{G}_2$ and $\mathcal{G}_3$ are 80, 100 and 120, respectively.
- The computation time is chosen as a random value with uniform distribution from 2 to 30.
- Each node has equal CPU utilization.

Each plot on the graphs shown in this section represents the average of a set of 100 independent simulations. Fig. 3 shows the worst case response time of global tasks for the case of utilization $\alpha = 0.5$, 0.7, 0.8 and 0.9. Under $\alpha = 0.5$, 0.7 and 0.8, the proposed algorithm is able to obtain a significant improvement compared to the background algorithm and the performance of the proposed algorithm is similar to that of the slack stealing algorithm. This is due to the fact that subtasks are assigned higher priorities than local tasks under low and medium load conditions. For $\mathcal{G}_3$, the proposed algorithm shows slightly better performance than the slack stealing algorithm. The reason is that the relative order of subtasks between nodes is changed. If $\mathcal{G}_{j,k-1}$ has lower priority than other subtasks of different global tasks and completes its computation too late, $\mathcal{G}_{j,k}$ will have the small slack and shorter deadline. Thus, $\mathcal{G}_{j,k}$ will be assigned a higher priority than other subtasks of different global tasks. On the contrary, in background and slack stealing algorithm, the order of subtasks is fixed for all nodes and the last subtask always has a lower priority and experiences longer response time. As a result, the proposed algorithm tends to reduce the variance of the worst case response times of global tasks. This property is best shown for the case of $\alpha = 0.9$ in Fig. 3(d). Under high load condition, subtasks in the proposed algorithm are assigned lower priorities and experience longer response time. However, the variance is much smaller than that of global tasks in the slack stealing algorithm.

7 Conclusion

In this paper we present a priority assignment algorithm to improve the end-to-end response time of a global task in a distributed real-time system. In the proposed scheme, we assign the highest possible priorities to subtasks without missing deadline of local tasks. Simulation results show that the performance of the proposed algorithm approaches that of the slack stealing algorithm when α is 0.5, 0.7 and 0.8. Although the worst case response time of global tasks increases compared with the slack stealing algorithm under high load($\alpha = 0.9$), the variance of the worst case response time is reduced. Since tasks are scheduled by a fixed priority scheduler, there is no additional run-time overhead. For the future research, we will apply the proposed scheme to an actual control system to visualize its impact on the system behavior.

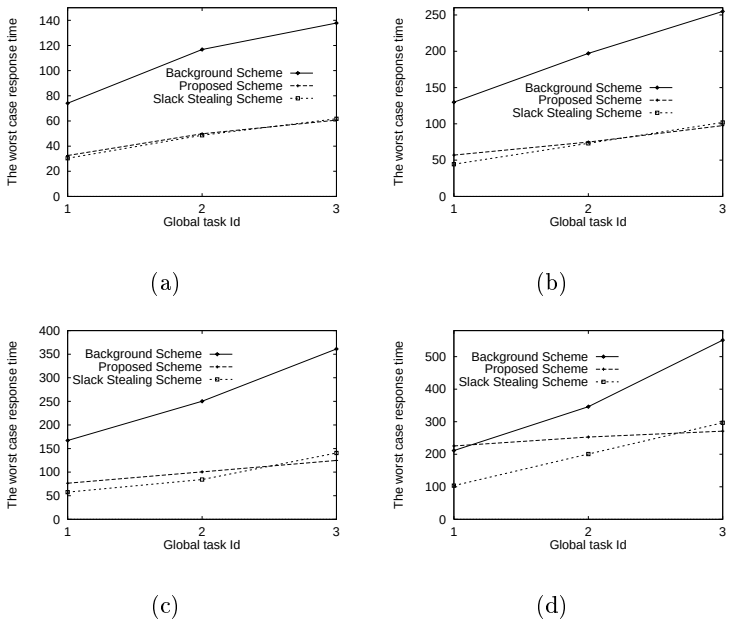

Fig. 3. The worst case response time of global tasks at (a) $\alpha = 0.5$, (b) $\alpha = 0.7$, (c) $\alpha = 0.8$, (d) $\alpha = 0.9$

References

1. N. C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical report, University of York, 1991.
2. Alan Burns. Preemptive priority-based scheduling: An appropriate engineering approach. In *Advances In Real-Time Systems*, pages 225–248. Prentice Hall, 1995.
3. B. Kao and H. Molina. Deadline assignment in a distributed soft real-time system. In *13th International Conference on Distributed Computing Systems*, 1993.
4. J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proceedings of the Real-Time Systems Symposium*, pages 201–209, 1990.
5. John P. Lehoczky and Sandra Ramos-Thuel. An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems. In *Proceedings of the Real-Time Systems Symposium*, pages 110–123, 1992.
6. J. Y. T. Leung and J. Whitehead. On the complexity fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4):237–250, 1982.
7. M. Saksena and S. Hong. An engineering approach to decomposing end-to-end delays on a distributed real-time system. In *Proceedings of the 4th International Workshop on Parallel and Distributed Real-Time Systems*, 1996.
8. Skvarcius and Robinson. *Discrete Mathematics with Computer Science Applications*. The Benjamin/Cummings Publishing Company, 1986.
9. B. Sprunt, L. Sha, and J. P. Lehoczky. Aperiodic task scheduling for hard real-time systems. *The Journal of Real-Time Systems*, 1:27–60, 1989.