

Double-loop Feedback-based Scheduling Approach for Distributed Real-Time Systems

Suzhen Lin and G. Manimaran

Department of Electrical and Computer Engineering
Iowa State University, Ames, IA 50011, USA
{linsz, gmani}@iastate.edu

Abstract. The use of feedback control techniques has been gaining importance in real-time scheduling as a means to provide predictable performance in the face of uncertain workload. In this paper, we propose and analyze a feedback scheduling algorithm, called *double-loop feedback scheduler*, for distributed real-time systems, whose objective is to keep the deadline miss ratio near the desired value and achieve high CPU utilization. This objective is achieved by an integrated design of a local and a global feedback scheduler. We provide the stability analysis of the double-loop system. We also carry out extensive simulation studies to evaluate the performance and stability of the proposed double-loop scheduler. Our studies show that the proposed scheduler achieves high CPU utilization with low miss ratio and stays in steady state after a step change in workload, characterized by change in actual execution time of tasks.

1 Introduction and Motivation

In real-time systems, the correctness of the system depends not only on the logical correctness of the result, but also on the time at which the results are produced [1][2]. Traditional real-time scheduling algorithms are based on estimations of the (pessimistic) worst-case execution time (*WCET*) of tasks. However, this will result in an extremely underutilized system. In many cases, it is preferable to base scheduling decisions on a more correctly estimated execution time (*ET*) and to be ready to deal with changes in execution times dynamically. This approach is especially preferable as it provides a firm performance guarantee in terms of deadline misses while achieving high throughput.

One of the very successful areas in addressing performance in the presence of uncertainty is control theory. The feedback strategy is useful primarily if uncertainty is present and the advantages toward performance far outweigh the associated complexity in implementation.

In dynamic real-time systems, three steps are involved in task management: Schedulability check (admission control), Scheduling, and Task execution. The scheduler is responsible for performing the first two steps and processor(s) are responsible for executing the tasks. The first step decides if the tasks can be admitted into the system, and the scheduling algorithm decides where (which processor) and when to execute the admitted tasks. Since the system resources are limited, some tasks may be rejected during the schedulability check. Even after tasks are admitted to the system for execution, there may be still some tasks that miss their deadlines due to the uncertainty in execution

times. *Task rejection ratio* (RR) is defined as the ratio of the number of tasks rejected by the admission controller to the number of tasks arrived in the system. *Deadline miss ratio* (MR) is defined as the ratio of the number of tasks that miss their deadlines to the number of tasks admitted by the admission controller. Ideally, one would prefer a low RR if possible, low MR , and high *CPU utilization* (U) in an overloaded system. U can be defined as $\frac{CPU_{busytime}}{CPU_{busytime}+CPU_{idletime}}$ in a certain time interval.

When the scheduler performs the admission test and scheduling, it uses task parameters, such as execution time and deadline. Among the parameters, the (actual) execution time of a task depends on conditional statements and data dependent loops that are influenced by the dynamics of the environment in which the system is operating. Thus, task execution time creates workload uncertainty in the system. Consider image processing (includes digital filtering for noise reduction and edge detection) [1] for object moving on a conveyor belt. Task execution time may change due to environmental factors such as lighting condition and object orientation. For example, if the light of the environment becomes dark, the image processing will take longer time, since more work needs to be done for noise reduction and edge detection.

There are three ways to deal with the uncertainty in execution time: (1) Schedule tasks based on *WCET* obtained through pessimistic static analysis. This will result in high RR (also low U), but zero/less MR ; (2) Schedule tasks based on best-case execution time (*BCET*) obtained through analysis based on optimistic assumptions. This will result in low RR , but may incur high MR . Thus, the task execution time introduces a trade-off between RR and MR . (3) *Schedule tasks based on an estimate of actual execution time (AET) obtained by using feedback control algorithm*. Unlike approaches (1) and (2), approach (3) has the potential to capture this trade-off in order to minimize both RR and MR , by making a good estimate of *AET*.

For convenience, we define *CPU utilization factor* (UF) as $(1 - \text{requested CPU utilization})$. The *requested CPU utilization* (U_r) is defined as the summation of *AET* over the corresponding period of admitted tasks. Ideally, UF should be close to 0. In the feedback control context, in order to achieve low MR and high U , we monitor MR and UF and feed them back to the controller, and regulate the two performances to desired values by adjusting appropriate parameters like *EET*. The reason that we do not choose RR as the regulated variable is because it depends not only on the estimation of execution times, but also on the number of tasks.

Scheduling in distributed systems involves two main components: (1) local scheduling which schedules tasks on a given node; (2) global scheduling which migrates tasks to a lightly loaded node if the current node is overloaded [2].

Our objective is to develop a feedback-based scheduling algorithm for distributed real-time systems (DRTS), with the goal of achieving low MR and high U . We propose a double-loop feedback control-based scheduling algorithm of which the local controller (inner loop) is associated with the local scheduler and the global controller (outer loop) is associated with the global scheduler. The local controller will be used to estimate the execution time of tasks and the global controller will be used to adjust the set points (desired values of the output) of the local system as a means to facilitate load balancing.

- Local scheduler: EDF, RMS, or Spring scheduling algorithm.
- Processor: executing tasks.
- MR , U and UF measurers: calculating the MR , U and UF . UF can be gained by calculating U_r . By measuring the tasks execution time in a past interval, we can calculate U_r .
- Feedback information collector: collecting the performances and feeding back to local and global controllers.
- Average Calculator: Calculating the average performances of the neighborhood.
- Global PID controller: PID control law is used.
- Global actuator: changing the set points for the local control system.
- Global scheduler: deciding which tasks and where to migrate based on load index.

We employ two controllers: *global controller (or distributed controller)* and *local controller* [9]. Each node in the distributed system has these two controllers. The global controller gets information from its neighbors and itself, and then gives outputs according to the control law it uses. The outputs of the global controller are used as set points for the local controller. In other words, the local controller and the local system are treated as the controlled system, controlled by the global controller. The global controller cooperates with the local controller to adjust the performances of the local node in terms of deciding the number of tasks to be rejected by the local system. The rejected tasks that have not missed their deadlines may be migrated to other nodes. That is, in some sense, the global controller is responsible to achieve load balance in the distributed system.

3.1 Feedback Control

System Variables: In real-time scheduling, we choose MR and UF as regulated variables and measured variables, since these are the metrics indicating how well the system is behaving and resources are being used. The set points are desired values for MR and UF . We ignore the disturbance in our discussion. The control variable is the estimation factor for the task execution time, since the change of task execution time may lead to poor performance if the estimated execution time remains unchanged.

Control Law: The control law used in the controllers is PID [10]. PID stands for Proportional, Integral and Derivative. Controllers are used to adjust EET to hold UF and MR at the set point. The set point is desired value of the system output. Error (the controller input) is defined as the difference between set point and measurement. The output of a controller changes in response to a change in measurement or set point, since the error changes when measurement or set point changes.

Controller Algorithms: We consider PI controller for the local controller and PI controller for the global controller.

Local System: Assume the task set is $\{T_1, T_2, \dots, T_i, \dots\}$. EET_i , $AvCET_i$, and $BCET_i$ are the estimated execution time, the average case execution time ($AvCET$) and the best case execution time of task T_i respectively. The set points for the local controller are MR_{LS} and UF_{LS} . UF and MR of the node are fed back to the controller periodically for every time interval T . So UF and MR are collected at regular intervals: $0, T, \dots, kT, \dots$. In our algorithm, we use an estimation factor (etf) to estimate the execution time of tasks, the estimation factor adjust the estimated execution time by

increasing or decreasing the execution time from the average case execution time. At instance k , this adjustment is shown in Equation 1.

$$(EET_i)_k = AvCET_i + etf_k(AvCET_i - BCET_i) \quad (1)$$

etf_k is the estimation factor at time instance k . When the system starts to work, we set the etf_0 value to be 0, that is, we use the average case execution time to perform the admission test and schedule tasks. Then we get the feedback performances (MR_k and UF_k , which are MR and UF at time instance k respectively); according to these information, we get the amount of change for the estimation factor (Δetf_k) and then change the estimation factor. This is shown in Equation 2 and 3.

$$\Delta etf_k = K_m(MR_{LS} - MR_{k-1}) - K_u(UF_{LS} - UF_{k-1}) \quad (2)$$

$$etf_k = etf_{k-1} - \Delta etf_k \quad (3)$$

In Equation 2, let K_m and K_u be positive values. If MR_{k-1} is greater than MR_{LS} , this means the estimation for the execution time is too small, hence MR is high. We have to increase etf_k in order to make MR small. $K_m(MR_{LS} - MR_{k-1})$ contributes a negative part to Δetf_k . According to Equation 3, this will contribute a positive part to etf_k . Thus, the feedback information of MR_{k-1} will lead to the increase of etf_k , so as to decrease MR . Similarly, if UF_{k-1} is greater than UF_{LS} , this means the estimation for the execution time is too large, CPU will allocate too much time for each task and U_r is low. We have to decrease etf_k in order to make UF close to zero. $-K_u(UF_{LS} - UF_{k-1})$ contributes a positive part to Δetf_k . According to Equation 3, this will contribute a negative part to etf_k . Thus, the feedback information of UF_{k-1} will lead to the decrease of etf_k , so as to decrease RR . For the case that MR_{k-1} is less than MR_{LS} or UF_{k-1} is less than UF_{LS} , the analysis is the same.

Global System: Compared with the local controller, the global controller acts slowly. It gets information from its neighbors and computes the set points for the local controller. At every time instance, the global controller gets MR and UF from its neighbors and itself, it uses the averages of MR s and UF s as the global set points. Assume these two averages are MR_A and UF_A , then the outputs of the global controller are given by Equation 4.

$$\begin{bmatrix} MR_{LS} \\ UF_{LS} \end{bmatrix}_k = \begin{bmatrix} MR_{LS} \\ UF_{LS} \end{bmatrix}_{k-1} + \begin{bmatrix} K_{11} & -K_{12} \\ -K_{21} & K_{22} \end{bmatrix} \left(\begin{bmatrix} MR_A \\ UF_A \end{bmatrix}_{k-1} - \begin{bmatrix} MR \\ UF \end{bmatrix}_{k-1} \right) \quad (4)$$

MR_{LS} considers the influence not only from MR_A but also from UF_A , so does UF_{LS} . So we will let K_{11} , K_{12} , K_{21} and K_{22} be positive values. These values are coefficients, which can be determined experimentally. According to Equation 4, we have $MR_{LSk} = MR_{LS(k-1)} + K_{11}(MR_{A(k-1)} - MR_{k-1}) - K_{12}(UF_{A(k-1)} - UF_{k-1})$. Consider if the current node's MR is less than MR_A of the whole distributed system, then the current node needs to increase MR set point. The increased part is: $K_{11}(MR_{A(k-1)} - MR_{k-1})$. At the same time, in order to increase MR of the system, the current node can decrease U_r , so that MR can be increased further. The increased part is: $-K_{12}(UF_{A(k-1)} - UF_{k-1})$. Since we only need to decrease UF when UF is greater than the average UF of the whole distributed system. So, this part is greater than zero in such situation. Other analysis is similar.

3.2 Stability Analysis

For discrete system, the sufficient and necessary condition for stability is that all the eigen values of the characteristic equation lie within the unit circle in Z plane (assuming no zeros and poles cancellation). Figure 2 is the block diagram for the local system.

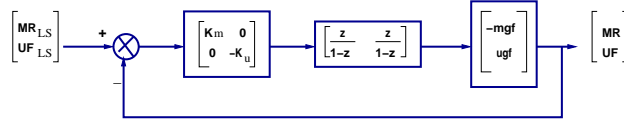


Fig. 2. Block diagram for the local system

The characteristic equation of the local control system is given in Equation 5. $G_L(z)$ and $G_K(z)$ are the transfer functions of the local system and controller respectively.

$$I + G_L(z)G_K(z) = 0 \quad (5)$$

Assume that mgf and ugf are the gains that map the estimated factor to MR and UF respectively. Then the transfer function of the local system is given in Equation 6.

$$G_L(z) = \begin{bmatrix} -mgf \\ ugf \end{bmatrix} \quad (6)$$

We can get the characteristic equation as shown in Equation 7.

$$\begin{bmatrix} 1 + \frac{z}{z-1}K_m mgf & \frac{z}{z-1}K_u mgf \\ \frac{z}{z-1}K_m ugf & 1 + \frac{z}{z-1}K_u ugf \end{bmatrix} = 0 \quad (7)$$

The eigen values are $z_{1,2} = 0$, $z_3 = \frac{1}{1+K_m mgf}$, $z_4 = \frac{1}{1+K_u ugf}$. Since $K_m mgf > 0$, $K_u ugf > 0$, all the eigen values lie within the unit circle. So, our local control scheduling system is stable.

In the double-loop control system, the inner loop will respond to changes much more quickly than the outer loop. So, when we consider the global controller, we can treat the local system as a model that has transfer function I (identity matrix). Then using PI controller, the analysis for the global controller will be the same as the local one. The block diagram for the double-loop system is shown in Figure 3.

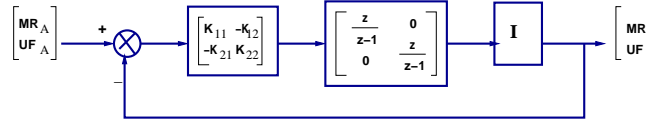


Fig. 3. Block diagram for the global system

Similarly, we can get the characteristic equation as shown in Equation 8.

$$\begin{bmatrix} 1 + \frac{z}{z-1}K_{11} & -\frac{z}{z-1}K_{12} \\ -\frac{z}{z-1}K_{21} & 1 + \frac{z}{z-1}K_{22} \end{bmatrix} = 0 \quad (8)$$

The eigen values are $z_{1,2} = 0$, $z_3 = \frac{1}{1+K_{11}}$, $z_4 = \frac{1}{1+K_{22}}$. Since $K_{11} > 0$, $K_{22} > 0$, all the eigen values lie within the unit circle. So, our double-loop control system is stable.

3.3 Scheduling Algorithms

In DRTS, task scheduling involves local task scheduling and global task scheduling. For local task scheduling, since the adjustment of execution time of tasks are performed by controller and actuator, the scheduler can schedule tasks by using any real-time scheduling algorithms such as RMS (Rate Monotonic Scheduling) [11], EDF (Earliest Deadline First) [11], Spring Scheduling Algorithm [12], etc. The global scheduler needs to migrate tasks from overloaded nodes to underloaded nodes. A good load index should be defined to precisely characterize the system state.

Load Index: We consider U , RR and MR to form the load index, which will be used to guide task migration in the distributed system. U can tell us if the CPU of the

node is used effectively. However, we also need to consider the incoming load, that is, the tasks submitted to the node, since these tasks will take the CPU time if they are admitted at the node. Thus, we consider two factors to form the load information. Besides, for nodes that have the same U and RR , if the computation times of tasks change, increase for example, then the load of the system will increase too. Since the increase of tasks' computation times will lead to the increase of not only U but also MR , we also need to consider MR . Thus large U , RR , or MR indicates high load of the node, and we can use the following linear combination of the three factors to get the load information L (load index) for each node:

$$L = \alpha U + \beta RR + \gamma MR \quad (9)$$

where α , β and γ are coefficients in the interval $[0,1]$ and $\alpha + \beta + \gamma = 1$. For example, if we set $\alpha = \frac{1}{3}$, $\beta = \frac{1}{3}$, and $\gamma = \frac{1}{3}$, this means we consider $\frac{1}{3}$ influence from each of U , RR , and MR . The choice of α , β and γ is application dependent.

Load Balancing: Each node maintains the load information of nodes in its neighborhood in load information table. When the load L is greater than a certain threshold value $L_{threshold}$ (overloaded), the node can migrate tasks to its neighbors based on the load information table. The load information is exchanged periodically. If some nodes have a large number of neighbors, we can only maintain $min(d, h)$ neighbors' load information. d is the number of neighbors, and h denotes the computing capability of a node with respect to the least compute-capable node whose h value is 1. Each node can have the same or different h value. For example, node i has $h = 2$, and node j has $h = 4$, this means the computing capability of node i is twice of that of node j . If d is greater than h , the node can choose h neighbors arbitrarily to maintain the load information.

Now, the remaining problem is how to allocate tasks to the neighbors according to the load information table. Our main idea is to allocate tasks to the node's neighbors who are underloaded. Assume that S is the set of the current node's neighbors and whose load index are less than $L_{threshold}$, and $i \in S$, we have Equation 10:

$$N_i = N \times \frac{L_{threshold} - L_{vi}}{|S| \times L_{threshold} - \sum_{j \in S} L_j} \quad (10)$$

where N is the total number of tasks to be migrated from an overloaded node of which N_i is the number of tasks migrated to node v_i , and $|S|$ is the number of nodes in S .

[9] proposed two logic network structures for task migration: hierarchical structure and neighborhood structure. The difference between these two structures is the way the global controller gives set points to local controller. The hierarchical structure does not scale well and the neighborhood structure scales better.

We notice that the two structures are logical network structures, which incur high complexity and overhead to maintain them. In contrast, our paper bases on the physical network structure, and each node contacts its neighbors to get MR , RR , and U . We do not need to maintain the logical structure, and hence less overhead. Besides, we allocate the number of migrated tasks according to neighbors' remaining ability, that is, neighbor whose load index is less than the threshold load index by a large value will get more migrated tasks than those whose load index is less than the threshold load index by a small value.

4 Simulation Studies

Since the local and global controllers work in different time scale, we study the performances of the local and global system separately. The simulation model is as follows:

- EDF algorithm is used for local scheduling.
- A task set with average $U_r > 1$ is generated at the beginning of every time interval T . Each task has a $WCET$, $BCET$, period, arrival time, and $AvCET$ with $AvCET = \frac{BCET+WCET}{2}$.
- The feedback is obtained for every T time units and is taken to be 200 in the simulation.
- To study the ability that the controller adapts the parameters, we use step load request in task execution time. We define a *load indicator*, L_{ind} , to indicate the load in terms of execution time of tasks. L_{ind} is equal to 1 when $AvCET$ is used. $L_{ind} < 1$ indicates underload, and $L_{ind} > 1$ indicates overload.
- We use UF and MR as set points. But due to the discrete property and the length of task execution times, the performances may not be exactly equal to the set points. So, we only require the performances to be close to the set points. We use $MR_{LS} = 0$ and $MR_{LS} = 0$ as set points values. The measured performances are UF and MR .

4.1 Simulation Studies – Local System

We study the performances of local system in two cases, one is when the task execution time increases and the other is when the execution time decreases. In our studies, we compare the performances of three approaches: (1) our feedback approach, (2) open-loop approach based on average computation time, and (3) open-loop approach based on worst-case computation time.

Task Execution Time Increases: In this case, at time instance 200 we give a step load request, AET of all tasks are increased by some value, that is, we let $L_{ind}=1.5$, then we observe the change of MR and UF of the system. Figure 4(a) shows the change of MR and U_r factor when the load indicator changes from 1 to 1.5 at time 200 for the three approaches. In the feedback approach, at time 400, the UF and MR measurers detect that $UF = -0.32$ and $MR = 0.48$. The controller calculates the error (difference between the set points and the feedback values) and begins to adjust the estimation factor to achieve good UF and MR values. From the figure, we see that at time 800, UF becomes stable, and the final value is 0.039. This means U is equal to $1 - 0.04 = 0.96$. MR becomes stable at time 600 and the final value is 0.0. This means no tasks miss deadline. Thus, we get a high U . In the open-loop approach based on the average computation time, UF stays at -0.32 and MR stays at 0.48 after the step load is applied. This means U is high (close to 1) since negative UF means the system is overloaded, but 48% tasks miss their deadlines. In the open-loop approach based on $WCET$, UF increases from 0.2 to 0.24 and stays at this value. MR stays at 0. This means U is 76%. Comparing the three approaches, we see that when the load increases, the feedback approach can achieve high U and low MR . The open-loop approach based on average computation time can achieve high U , but MR is also high. The open-loop approach based on $WCET$ can achieve low MR , but U is low.

Task Execution Time Decreases: In this case, we change L_{ind} from 1 to 0.7. Then we observe the change of MR and UF of the system. Figure 4(b) shows the change of MR and UF when the load indicator changes from 1 to 0.7 at time 200 for the three approaches. Comparing these three approaches, we see that when the load decreases, the feedback approach offers high U and low MR . The other two approaches offer low MR , but U is lower than that of feedback approach.

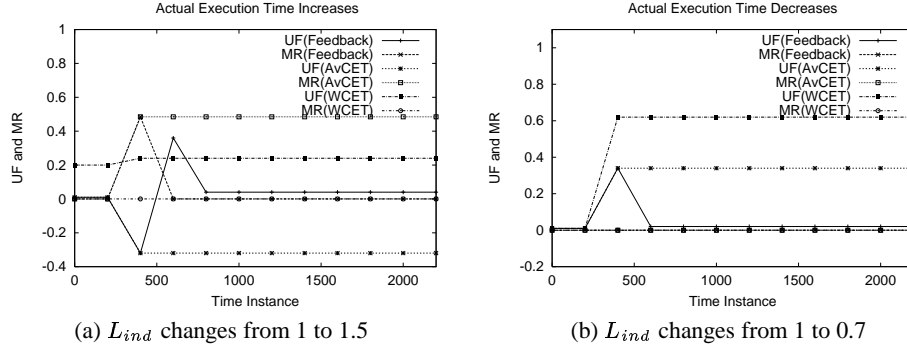


Fig. 4. UF and MR of local system

So the PI controller can adjust the *etf* of tasks according to the load change (computation time change) and maintain U and MR near the desired values.

4.2 Simulation Studies — Global System

Since the local system acts very quickly compared with the global controller. We can treat the transfer function of the local system as an Identity matrix, that is, the outputs of the local system are approximately equal to the set points from the global system.

Our simulation shows that the global controller can adjust MR and UF for the local system and achieves load balancing for the distributed system.

Task Execution Time Increases: Figure 5(a) shows the changes of UF and MR when AET of tasks of the local system increases at time 2000. This increase causes U_r to be higher than the new average U_r , that is, UF is less than the new average UF . The increase also causes MR to be higher than the new average MR . The global controller pulls MR and UF to new average values due to the feedback control.

Task Execution Time Decreases: Figure 5(b) shows the changes of UF and MR when AET of the local system decreases at time 2000. This decrease causes U_r to be lower than the new average U_r of the distributed system. This means UF is increased to be larger than the new average UF of the distributed system. The global controller pulls UF to new average value due to the feedback control.

Therefore, the global controllers can adjust the set points of the local systems and let the systems become stable quickly by working with local controllers.

5 Conclusions

In this paper, a double-loop feedback scheduling approach is proposed for distributed real-time systems. By using feedback control theory, the inner loop feedback control

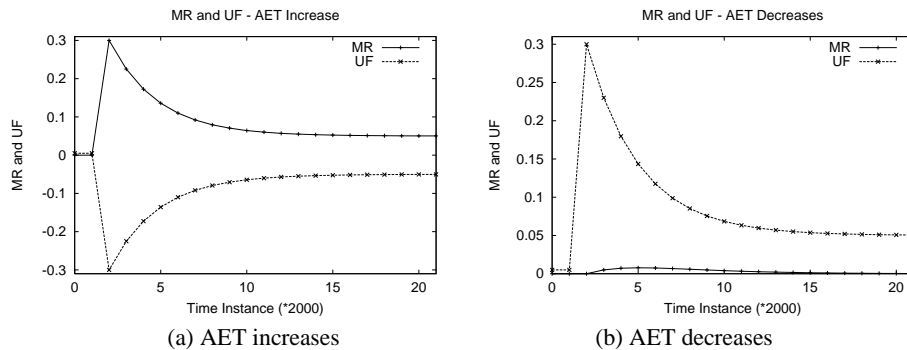


Fig. 5. MR and UF - Actual execution time of tasks changes

system adjusts the estimated execution time of tasks to achieve desired U and MR for the local system, and the outer loop control system adjusts the set points for each local system. Then we analyzed the stability of the double-loop feedback-based scheduling system in Z domain. We also proposed a novel load index, which considers MR , RR , and U . The performance and stability of the proposed double-loop scheduler are evaluated through simulation studies. Our studies show that good performances can be achieved. There are several possible avenues for further work in this emerging area of research, which includes the following: (1) feedback-based scheduling algorithm that takes into account network link load in addition to CPU load for global scheduling, and (2) feedback-based fault-tolerant scheduling algorithm that monitors system fault rate and performs redundancy management accordingly.

References

1. G. C. Buttazzo, "Hard Real-Time Computing Systems", *Kluwer Academic Publisher*, 1997.
2. C. Siva Ram Murthy and G. Manimaran, "Resource Management in Real-Time Systems and Networks", *MIT Press*, April 2001.
3. C. Lu, J. A. Stankovic, G. Tao, and S.H. Son, "Design and evaluation of feedback control EDF scheduling algorithm", In Proc. *IEEE RTSS*, pp.56-67 1999.
4. J. A. Stankovic, Chenyang Lu, S. H. Son, and G. Tao "The case for feedback control real-time scheduling", In Proc. *Euromicro Conference on Real-Time Systems*, pp.11-20, 1999.
5. D. R. Sahoo, S. Swaminathan, R. Al-Omari, M. V. Salapaka, G. Manimaran, and A. K. Somani, "Feedback control for real-time scheduling", In Proc. *American Controls Conference*, vol.2, pp.1254-1259, 2002.
6. R. Al-Omari, G. Manimaran, M. V. Salapaka, and A. K. Somani, "New algorithms for open-loop and closed-loop scheduling of real-time tasks based on execution time estimation", In Proc. *IEEE IPDPS*, 2003.
7. D. R. Alexander, D. A. Lawrence and L. R. Welch, "Feedback control resource management using a posteriori workload characterizations", In Proc. *IEEE Conference on Decision and Control*, vol.3, pp.2222-2227, 2000.
8. B. Ravindran, P. Kachroo, and T. Hegazy, "Adaptive resource management in asynchronous real-time distributed systems using feedback control functions", In Proc. *Intl. Symposium on Autonomous Decentralized Systems*, pp.39-46, 2001.
9. J. A. Stankovic, T. He, T. F. Abdelzaher, M. Marley, G. Tao, S.H.Son, and C.Lu, "Feedback control scheduling in distributed systems", In Proc. *IEEE RTSS*, pp.59-70, 2001.
10. K. Ogata, "Modern Control Engineering", *Prentice Hall*, Upper Saddle River, New Jersey, 2002.
11. C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment", *Journal of ACM*, vol.20, no.1, pp.46-61, January 1973.
12. J. A. Stankovic and K. Ramaritham, "The Spring Kernel: a new paradigm for real-time systems", *IEEE Software*, vol.8, no.3, pp.62-72, May 1991.