End-to-End Utilization Control in Distributed Real-Time Systems

Chenyang Lu Xiaorui Wang
Department of Computer Science and Engineering
Washington University in St.Louis
St. Louis, MO 63130
{lu, wang}@cse.wustl.edu

Xenofon Koutsoukos
Department of Electrical Engineering and Computer Science
Vanderbilt University
Nashville, TN 37235
xenofon.koutsoukos@vanderbilt.edu

Abstract

An increasing number of distributed real-time systems face the critical challenge of providing end-to-end Quality of Service (QoS) guarantees in open and unpredictable environments. In particular, such systems often need to guarantee the CPU utilization on multiple processors in order to achieve overload protection and meet end-to-end deadlines while task execution times are unpredictable. While the recently developed feedback control real-time scheduling algorithms have shown promise, they cannot handle the common end-to-end task model in distributed systems where each task is comprised of a chain of subtasks distributed on multiple processors. This paper presents the End-to-end Utilization CONtrol (EUCON) algorithm that features a distributed feedback loop that dynamically enforces desired CPU utilization bounds on multiple processors based on online performance measurements EUCON is based on a model predictive control approach that models the utilization control problem on a distributed platform as a multi-variable constrained optimization problem. A multi-input-multi-output model predictive controller is designed based on a difference equation model that describes the dynamic behavior of distributed real-time systems. Both control theoretic analysis and simulations demonstrate that EUCON can provide robust utilization guarantees even when task execution times deviate from the estimation or vary significantly at run-time.

Index terms*—real-time and embedded systems, feedback control real-time scheduling, distributed systems, end-to-end task, Quality of Service*

1. Introduction

In recent years, a category of performance-critical distributed systems executing in open and unpredictable environment has been rapidly growing [2]. Examples of such systems include distributed real-time embedded (DRE) systems such as avionics mission computing, autonomous aerial surveillance, disaster recovery systems, and multi-tier E-business servers such as on-line trading servers. A key challenge in developing such systems is providing critical Quality of Service (QoS) guarantees while the workload and the underlying platform cannot be accurately characterized *a priori*. For example, the execution times of visual tracking applications can vary significantly as a function of the number of potential targets in a set of received camera images. Similarly, the resource requirements and the arrival rate of service requests in an on-line trading server can fluctuate dramatically. However, QoS guarantees are required in these systems despite their unpredictability. In particular, such systems often need to guarantee the CPU utilization on multiple processors in order to achieve overload protection and meet end-to-end deadlines. Failure to meet critical QoS guarantees such as CPU utilization constraints may result in loss of customers, financial damage, liability violations, or mission failures.

At the same time, modern DRE systems increasingly rely on middleware (e.g., Real-Time CORBA [13]) to meet QoS requirements on Common-Off-The-Shelf (COTS) platforms. A key benefit of deploying applications on middleware is achieving portability across different platforms so that a same application does not need to be re-implemented for every different platform. To achieve the same level of portability to QoS-critical applications, however, DRE middleware must support *QoS portability* [2][11] in addition to functional portability. A DRE middleware should allow applications be deployed on different platforms with the same critical QoS guarantees without the need for manual performance tuning. QoS portability requires the middleware to provide QoS guarantees without accurate knowledge about the underlying platform.

These new challenges require a paradigm shift from classical real-time computing that relies on accurate characterization of workloads and platform. In recent years, control theoretic approaches that we call *QoS control* have shown promise in providing robust QoS guarantees in unpredictable environments. While existing real-time scheduling approaches are concerned with statically assured avoidance of undesirable effects such as overload and deadline misses, the QoS control approach handles such effects dynamically via on-line performance feedback loops.

Existing work on QoS control has focused on providing guarantees on a *single* processor based on the assumption that tasks on different processors are independent from each other. Unfortunately, solutions for a single processor are not applicable to distributed systems that employ the *end-to-end task model* [7][19]. In such systems, a task is comprised of a chain of subtasks executing on different processors. The execution of a task involves the execution of multiple subtasks under precedence constraints. Since the end-to-end task model is the dominant execution model in DRE systems and multi-tier E-

Business server clusters, it is important to extend the QoS control framework to end-to-end tasks. QoS control of end-to-end tasks on a distributed platform introduces several new research challenges.

- QoS control in distributed systems is a *multi-input-multi-output* (*MIMO*) control problem where the system performance on multiple processors must be guaranteed simultaneously.
- The MIMO control problem in distributed systems is complicated by the fact that the performance on different processors is *coupled* to each other due to the correlation among subtasks belonging to the same task. Changing the rate of an end-to-end task will affect the utilization of all the processors where its subtasks are located. Hence the CPU utilization of a processor cannot be controlled independently.
- QoS control is often subject to *constraints* in distributed systems. Examples include desired bounds on CPU utilizations and limits on acceptable task rates.

As a step toward QoS control for the end-to-end task model, this paper proposes the *End-to-end Utilization CONtrol* (*EUCON*) algorithm. EUCON can guarantee desired CPU utilization in distributed systems even when task execution times vary significantly from the estimated ones. Furthermore, it can provide robust guarantees on end-to-end deadlines in DRE systems. The primary contributions of this paper are three-fold.

- Development of a *Model Predictive Control* (MPC) approach to QoS guarantees in DRE systems,
- Derivation of a dynamic model that captures the coupling among processors and constraints in DRE systems executing end-to-end tasks, and
- Design and control analysis of a distributed MIMO feedback control loop in EUCON that provide robust utilization guarantees when task execution times deviate from the estimated ones and vary significantly at run-time.

2. Related Work

Traditional approaches for handling end-to-end tasks are based on the end-to-end task model [19] or distributed priority ceiling [16]. Both are open-loop approaches based on schedulability analysis that rely on accurate knowledge about worst-case execution times. When task execution times are highly unpredictable, such open-loop approaches may severely underutilize the system. An approach for dealing with unpredictable task execution times is resource reclaiming [4][15]. A drawback of existing resource reclaiming techniques is that it often requires modifications to specific scheduling algorithms in operating systems, which is often undesirable in COTS platforms. In contrast, the feedback control approach adopted in this paper can be easily implemented at the middleware layer on top of COTS platforms.

The control theoretic approach has been applied to various computing and networking systems. A survey of feedback performance control for software services is presented in [2]. Recent research on applying control theory to real-time scheduling and utilization control is directly related to this paper. Steere, et al., developed a feedback scheduler [18] that coordinated the CPU allocation to consumer and supplier threads, to guarantee the fill level of buffers. Abeni, et al., presented control analysis of a reservation-based feedback scheduler in [3]. In [1], a feedback-control-based admission controller was designed to guarantee desired CPU utilization on an Apache server. In [10] a Feedback Control real-time Scheduling (FCS) framework and three FCS algorithms were developed to provide deadline miss ratio and CPU utilization guarantees for real-time applications with unknown task execution times. The FCS algorithms have been integrated with an Object Request Broker middleware [11]. All the aforementioned projects focused on controlling the performance of a *single* processor. In addition, their control designs are based on linear control techniques such as Proportional-Integral-Derivative (PID) control. This control design cannot be easily extended to end-to-end utilization control due to the coupling among multiple processors and practical constraints in DRE systems. FCS has been extended to handle distributed systems [17]. However, this work did not address end-to-end tasks. Instead, it assumed tasks on different processors were *independent* from each other.

3. Problem Formulation

In this section, we formulate the end-to-end utilization control problem in the context of DRE systems.

3.1. A Flexible End-to-End Task Model

A system is comprised of m end-to-end periodic tasks $\{T_i \mid 1 \le i \le m\}$ executing on n processors $\{P_i \mid 1 \le i \le n\}$. Task T_i is composed of a chain of subtasks $\{T_{ij} \mid 1 \le j \le n_i\}$ that may be allocated to multiple processors. A subtask T_{ij} $(1 < j \le n_i)$ cannot be released for execution until its predecessor T_{ij-1} is completed. We assume that a non-greedy synchronization protocol (e.g., release guard [19]) is used to enforce the precedence constraints between subsequent subtasks. Hence each subtask T_{ij} of a periodic task T_i is also periodic and shares the same rate as T_i [19]. Each task T_i is subject to an end-to-end relative deadline related to its period. In this work, we assume task deadlines are soft, i.e., applications can tolerate a small number of deadline misses.

Each subtask T_{ij} has an *estimated* execution time c_{ij} at design time. However, the actual execution time of a task may be significantly different from its estimation and may vary at run time.

We assume that the rate of T_i can be dynamically adjusted within a range $[R_{min,i}, R_{max,i}]$. Earlier research has shown that many DRE applications (e.g., digital feedback control [5][14], sensor data display, and video streaming) have flexible task

rates that can be adjusted without causing application failure. A task running at a higher rate contributes a higher *value* to the application at the cost of higher CPU utilization. Rate adjustment is an example of an adaptation mechanism that can be used to control CPU utilization. Other adaptation mechanisms such as admission control and task reallocation may also be incorporated into the QoS control framework.

3.2. Problem Formulation

Before formulating the end-to-end utilization control problem, we first introduce several notations.

- T_s: The sampling period.
- $u_i(k)$: The *CPU utilization* (or *utilization* for simplicity) of processor P_i in the k^{th} sampling period, i.e., the fraction of time that P_i is not idle during time interval $[(k\text{-}1)T_s, kT_s]$.
- B_i: The *utilization set point* of processor P_i. B_i is the desired utilization of P_i specified by the user.
- $r_i(k)$: The invocation rate of task T_i in the $(k+1)^{st}$ sampling period. In general, the sampling period T_s is selected so that multiple instances of each task may be invoked during a sampling period.
- w_i: The weight of processor P_i. A higher weight w_i assigns higher importance to controlling the utilization of P_i.

The end-to-end utilization control problem can be formulated as a constrained optimization problem. The goal is to minimize the difference between the utilization set point and the utilization

$$\min_{\{r_j(k)|1\le j\le n\}} \sum_{i=1}^{n} w_i (B_i - u_i(k))^2$$

subject to two sets of constraints:

$$u_i(k) \le B_i \ (1 \le i \le n) \quad (1)$$

$$R_{min,i} \le r_i(k) \le R_{max,i} \ (1 \le i \le m) \quad (2)$$

The utilization constraints (1) ensure that no processor exceeds its utilization set point. At the same time, the optimization goal avoids underutilizing the system by making the utilization of each processor as close to its set point as possible. The latter is important because CPU underutilization usually causes poor system performance. In our task model underutilization leads to low task rates, which corresponds to poor application performance such as low quality video or higher control cost in digital control systems [14].

It should be noted that, due to the dynamic nature of the workload, EUCON can only guarantee that the utilization of each processor P_i always *converge* to a value no higher than B_i. This *convergence guarantee* lies between a hard guarantee and a soft guarantee on average system performance [2].

3.3. Applications

EUCON provides a powerful technique for utilization control in a broad range of QoS-critical systems.

- *End-to-end real-time scheduling*: In DRE systems, real-time tasks must meet their end-to-end deadlines. In the end-to-end scheduling approach [7][19], the deadline of an end-to-end task is divided into subdeadlines of its subtasks [7], and the problem of meeting the deadline is transformed to the problem of meeting the subdeadline of each subtask. A well known approach for guaranteeing the subdeadlines on a processor is by enforcing the *schedulable utilization bound*. The subdeadlines of all the subtasks on a processor are guaranteed if the utilization of the processor remains below the schedulable utilization bound of its subtasks. For EUCON to guarantee end-to-end deadlines, a user should specify the utilization set point of each processor to no higher than its schedulable utilization bound. Existing real-time scheduling theory has established various schedulable utilization bounds for different task models (e.g., [8][9]).
- *QoS portability*: EUCON can also be implemented in DRE middleware to support *QoS portability* [11]. When an application is deployed on a faster platform, the task rates will be automatically increased to take advantage of the extra resource. On the other hand, when an application is deployed to a slower platform, task rates will be automatically reduced to maintain the same CPU utilizations guarantees. EUCON's self-tuning capability can significantly reduce cost of porting real-time applications across platforms.
- *Overload protection*: Most QoS-critical systems (e.g., E-business servers) desire to avoid saturation of processors, which may cause system crash or severe service degradation [1]. In COTS operating systems that support real-time priorities, high utilization by real-time threads may cause kernel starvation [11]. EUCON allows a user to enforce desired utilization bounds for all the processors in a distributed system. Moreover, the utilization set point can be changed online. For example, a user may lower the utilization set point on a particular processor in anticipation of future workload, and EUCON will dynamically readjust task rates to enforce the new set point.

DRE systems span a wide spectrum in terms of scale and network support. In this paper, we focus on small-scale DRE platforms (e.g., computing clusters) each composed of several processors connected through a high speed communication interface (*e.g.*, a VME backplane or a fast Ethernet switch). Many existing DRE systems such as avionics systems, shipboard computing, and process control systems fall into this category. A centralized QoS control architecture (with appropriate fault-tolerance support) is usually sufficient to this class of DRE systems. Decentralized control for larger-scale systems is part of our future work.

4. Overview of EUCON

EUCON is an adaptive algorithm that features a MIMO feedback control loop (see Figure 1) that dynamically adjusts task rates to enforce the utilization set points. The DRE system is controlled by a centralized MIMO controller. The controller may be located on a separate processor, or share a processor with some applications. EUCON must be scheduled as the highest-priority task in order to effectively control utilization under overload conditions. Each processor has a *utilization monitor* and a *rate modulator*. A separate TCP connection (called *feedback lane* in [11]) connects the controller with the pair of utilization monitor and rate modulator on each processor. The user inputs to the controller include the utilization set points, $\boldsymbol{B} = [B_1 \ \ldots \ B_n]^T$ and the rate constraints on each task. The *controlled variables* are the utilization of all processors, $\boldsymbol{u}(k) = [u_1(k) \ \ldots \ u_n(k)]^T$. The *control inputs* from the controller are the change to task rates $\Delta\boldsymbol{r}(k) = [\Delta r_1(k) \ \ldots \ \Delta r_m(k)]^T$, where $\Delta r_i(k) = r_i(k) - r_i(k\text{-}1)$ $(1 \le i \le m)$. The following feedback control loops are invoked in the end of every sampling period:

1. The utilization monitor on each processor sends the utilization $u_i(k)$ in the last sampling period to the controller through its feedback lane.
2. The controller collects the utilization vector $\boldsymbol{u}(k)$, computes $\Delta\boldsymbol{r}(k)$, and sends the new task rates to the rate modulator on each processor through its feedback lane.
3. The rate modulator on each processor changes the task rate according to the input from the controller.

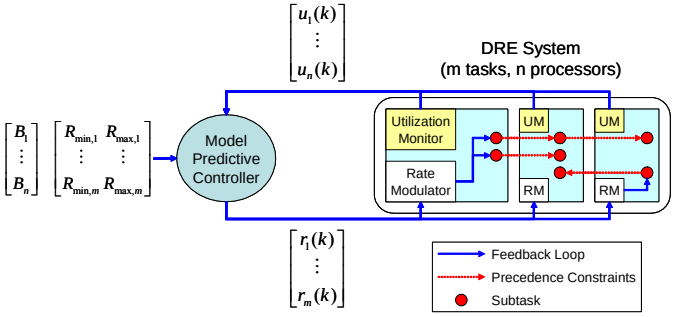


Figure 1. The MIMO feedback control loop

Since the core of EUCON is the controller, we will focus on the design of the controller in the rest of the paper. The design of the other components is similar to FCS/nORB [11], a feedback control scheduling service on an Object Request Broker middleware.

5. Dynamic Model of End-to-End Tasks

Following a control theoretic methodology, we must establish a dynamic model that characterizes the relationship between the control input $\Delta\boldsymbol{r}(k)$ and the controlled variable $\boldsymbol{u}(k)$.

First, we model the utilization $u_i(k)$ of one processor P_i. Let $\Delta r_j(k)$ denote the change to task rate, $\Delta r_j(k) = r_j(k) - r_j(k\text{-}1)$. We define the *estimated change to utilization*, $\Delta b_i(k)$, as

$$\Delta b_i(k) = \sum_{T_{jl} \in S_i} c_{jl} \Delta r_j(k) \tag{3}$$

where S_i represents the set of subtasks located at processor P_i. Note $\Delta b_i(k)$ is based on the estimated execution time. Since the actual execution times may be different from their estimation, we model the utilization $u(k)$ as

$$u_i(k) = u_i(k\text{-}1) + g_i \Delta b_i(k\text{-}1) \tag{4}$$

where the *utilization gain* g_i represents the ratio between the change to the *actual* utilization and its estimation $\Delta b_i(k\text{-}1)$. For example, $g_i = 2$ means that the actual change to utilization is twice of the estimated change. Note that the exact value of g_i is *unknown* due to the unpredictability of subtasks' execution times. Equation (4) models a single processor. We now model all the processors in the system. A system with m processors is described by the following MIMO model.

$$\boldsymbol{u}(k) = \boldsymbol{u}(k\text{-}1) + \boldsymbol{G}\Delta\boldsymbol{b}(k\text{-}1) \tag{5}$$

where $\Delta\boldsymbol{b}(k)$ is a vector including the estimated change to utilization of each processor, and $\boldsymbol{G}$ is a diagonal matrix where $g_{ii} = g_i$ $(1 \le i \le n$ and $g_{ij} = 0$ $(i \ne j)$.

The relationship between the utilization and task rates is characterized as follows.

$$\Delta\boldsymbol{b}(k) = \boldsymbol{F}\Delta\boldsymbol{r}(k) \tag{6}$$

The *subtask allocation matrix*, $\boldsymbol{F}$, is an $n \times m$-order matrix, where $f_{ij} = c_{jl}$ if a subtask T_{jl} of task T_j is allocated to processor i, and $f_{ij} = 0$ if no subtask of task T_j is allocated to processor i. Note that $\boldsymbol{F}$ captures the coupling among processors due to end-to-end tasks. Equations (5-6) give a dynamic model of a distributed system with m tasks and n processors.

Example: Suppose a system has two processors and three tasks. T_1 has only one subtask T_{11} on processor P_1. T_2 has two subtasks T_{21} and T_{22} on processors P_1 and P_2, respectively. T_3 has one subtask T_{31} allocated to processors P_2. We have

$$u(k) = \begin{bmatrix} u_1(k) \\ u_2(k) \end{bmatrix}, \quad G = \begin{bmatrix} g_1 & 0 \\ 0 & g_2 \end{bmatrix}, \quad \Delta b(k) = \begin{bmatrix} \Delta b_1(k) \\ \Delta b_2(k) \end{bmatrix},$$

$$F = \begin{bmatrix} c_{11} & c_{21} & 0 \\ 0 & c_{22} & c_{31} \end{bmatrix}, \quad \Delta r(k) = \begin{bmatrix} \Delta r_1(k) \\ \Delta r_2(k) \\ \Delta r_3(k) \end{bmatrix}.$$

From (5-6), the system model is

$$u_1(k) = u_1(k-1) + g_1(c_{11}\Delta r_1(k) + c_{21}\Delta r_2(k))$$
$$u_2(k) = u_2(k-1) + g_2(c_{22}\Delta r_2(k) + c_{31}\Delta r_3(k))$$

6. Model Predictive Controller

6.1. Design of a Model Predictive Controller

Based on the system model, a MIMO controller can be designed to guarantee the utilization set points on multiple processors. The PID control approach adopted in earlier works on feedback control real-time scheduling [10][17] is not suitable for DRE systems due to the coupling among multiple processors and the constraints. To solve this control problem, we adopt a *Model Predictive Control (MPC)* [12] approach. MPC is an advanced control technique used extensively in industrial process control applications. Its major advantage is that it can deal with coupled MIMO control problems with constraints on the plant and the actuators. This characteristic makes MPC very suitable for end-to-end utilization control in DRE systems that can be represented by MIMO system models under a set of constraints. The basic idea of MPC is to optimize an appropriate cost function defined over a time interval in the future. The controller employs a model of the system which is used to predict the behavior over P sampling periods called the *prediction horizon.* The control objective is to select an input trajectory that minimizes the *cost* while satisfying the constraints. An input trajectory includes the control inputs in the following M sampling periods, e.g., $\Delta \boldsymbol{r}(k)$, $\Delta \boldsymbol{r}(k+1|k)$, **…** $\Delta \boldsymbol{r}(k+M-1|k)$, where M is called the *control horizon*. Once the input trajectory is computed, only the first element ($\Delta \boldsymbol{r}(k)$) is applied as the input signal to the system. In the end of the next sampling period, the prediction horizon slides one sampling period and the input is computed again as a solution to a constrained optimization problem. MPC combines performance prediction, optimization, constraint satisfaction, and feedback control into a single algorithm. Details of MPC can be found in [12].

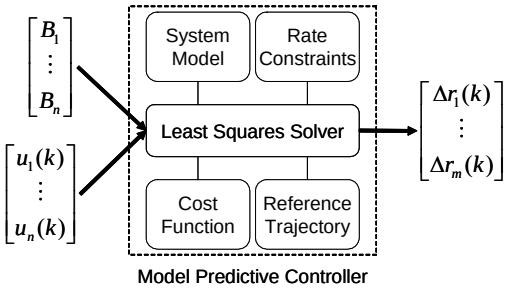


Figure 2. The model predictive controller

We now design a controller for EUCON. As illustrated in Figure 2, our model predictive controller includes a least squares solver, a cost function, a reference trajectory, and an approximate system model under the rate constraints. In the end of every sampling period, the controller computes the control input $\Delta \boldsymbol{r}(k)$ that minimizes the cost function under the utilization and rate constraints based on an approximate system model. We now describe each component in more details.

The cost function to be minimized by our controller is

$$V(k) = \sum_{i=1}^{P} \left\| u(k+i \mid k) - ref(k+i \mid k) \right\|_{Q(i)}^2 + \sum_{i=0}^{M-1} \left\| \Delta r(k+i \mid k) - \Delta r(k+i-1 \mid k) \right\|_{R(i)}^2 \quad (7)$$

where P is the prediction horizon, M is the control horizon, $\boldsymbol{Q}(i)$ is the *tracking error weight*, and $\boldsymbol{R}(i)$ is the *control penalty weight*. The notation $\boldsymbol{x}(k+i|k)$ means that the vector signal $\boldsymbol{x}$ depends on the conditions at time k.

The first term in the cost function represents the tracking error, i.e., the difference between the utilization vector $\boldsymbol{u}(k+i|k)$ and a *reference trajectory* $\boldsymbol{ref}(k+i|k)$. The reference trajectory defines an ideal trajectory along which the utilization vector $\boldsymbol{u}(k+i|k)$ should change from the current utilizations $\boldsymbol{u}(k)$ to the utilization set points $\boldsymbol{B}$. Our controller is designed to track the following exponential reference trajectory so that the closed-loop system will behave as a linear system.

$$ref(k+i \mid k) = B - e^{-\frac{T_s i}{T_{ref}}} (B - u(k)) \quad (8)$$

where T_{ref} is the time constant that specifies the speed of system response. By minimizing the tracking error, the closed-loop system will converge to the utilization set point if the system is stable. A higher T_{ref} causes the system to converge faster to the set points. The weight matrix $\boldsymbol{Q}(i)$ can be tuned to represent preferences between processors. For example, we can assign a higher weight to a processor if it executes more important applications. The second term in the cost function represents the control penalty. The control penalty term ensures that the controller will minimize the changes in the control input.

We have established a system model for DRE systems in Section 5. However, the model cannot be directly used by the controller because the system gains $\boldsymbol{G}$ are unknown. Therefore the controller must use an approximate model. Our controller assumes $\boldsymbol{G} = [1 \ \dots \ 1]^T$ in (5), i.e., the controller assumes the actual utilization will be the same as the utilization predicted based on estimated ones. Hence our controller solves the constrained optimization based on an approximate system model described by (6) and

$$\boldsymbol{u}(k) = \boldsymbol{u}(k-1) + \Delta \boldsymbol{b}(k-1) \quad (9)$$

This approximate model may behave differently from the real system. However, as we prove in Section 6.2, the closed loop system under our controller can still maintain stability and guarantee desired utilization set points as long as $\boldsymbol{G}$ is within a certain range. Furthermore, this range can be established using stability analysis.

The controller must minimize the cost function (7) under the utilization and rate constraints (1-2) based on the approximate system model described by (6) and (9). This constrained optimization problem can be easily transformed to a standard constrained *least-squares* problem [12] (the transformation is not shown due to space limitations). The control-

ler implements a standard least-squares solver to solve the optimization problem on-line. In our simulator, we implement the controller based on the `lsqlin` solver in Matlab. `lsqlin` uses an active set method similar to that described in [6]. The computational complexity of `lsqlin` is polynomial to the product of the number of tasks, the number of processors, and the control and prediction horizons. Therefore, while our controller is capable of handling small to medium scale systems which are the focus of this paper, more efficient control algorithm may be needed in scale to larger scale systems.

6.2. Stability Analysis

In MPC, a system is called *stable* iff for any initial condition it will converge to the equilibrium point [12]. In our case, the equilibrium point of the closed-loop system is the vector of utilization set points $\boldsymbol{B}$. Hence a stable DRE system guarantees that the utilization of every processor converges to its set point. We now outline a general approach for analyzing the stability for a DRE system controlled by our controller.

1. Derive the control inputs $\Delta\boldsymbol{r}(k)$ that minimize the cost function based on the *approximate* system model described by (6) and (9).
2. Derive the closed-loop system model by substituting the derived control inputs $\Delta\boldsymbol{r}(k)$ into the *actual* system model described by (5-6). The closed-loop system model is in the form
   $$\boldsymbol{u}(k) = \boldsymbol{A}\boldsymbol{u}(k\text{-}1) + \boldsymbol{C} \quad (10)$$
   where $\boldsymbol{A}$ is a matrix whose eigenvalues depend on the utilization gains $\{g_i \mid 1 \le i \le n\}$.
3. Derive the stability condition of the closed-loop system described by (10). According to control theory, the closed-loop system is stable if all the eigenvalues of matrix $\boldsymbol{A}$ locate inside the unit circle in the complex space. Solving this stability condition will give the range of g_i ($1 \le i \le n$) where the system will guarantee stability.

Example: We now apply the stability analysis approach to the example system described in the end of Section 5. The system has 3 tasks and 2 processors. We set the prediction horizon $P = 2$ and the control horizon $M = 1$. According to the MPC theory, the system is also stable with any longer prediction horizon and control horizon if it is stable with shorter horizons. The time constant of the reference trajectory is $T_{ref}/T_s = 4$. The weights on all terms are 1. The cost function can be transformed to the following formula in scalar form

$$V(k) = \sum_{j=1}^{2}\sum_{i=1}^{2}(u_j(k+i\mid k) - ref_j(k+i\mid k))^2 + \sum_{j=1}^{3}(\Delta r_j(k) - \Delta r_j(k-1))^2 \quad (11)$$

Substitute the model parameters to (6) and (9), we have

$$\begin{bmatrix} u_1(k+1) \\ u_2(k+1) \\ u_1(k+2) \\ u_2(k+2) \end{bmatrix} = \begin{bmatrix} u_1(k) \\ u_2(k) \\ u_1(k+1) \\ u_2(k+1) \end{bmatrix} + \begin{bmatrix} c_{11} & c_{21} & 0 \\ 0 & c_{22} & c_{31} \\ c_{11} & c_{21} & 0 \\ 0 & c_{22} & c_{31} \end{bmatrix} \begin{bmatrix} \Delta r_1(k) \\ \Delta r_2(k) \\ \Delta r_3(k) \end{bmatrix} \quad (12)$$

Substitute (12) and the reference trajectory in (7) to (11), the cost function becomes a function of $\Delta\boldsymbol{r}(k)$. We then derive the control input vector $\Delta\boldsymbol{r}(k)$ that minimize the cost function through partial differentiation.

Following Step 2, we establish the closed-loop model by substituting $\Delta\boldsymbol{r}(k)$ derived in the last step into the actual system model (5-6). The closed-loop model is a function of the system gains (g_1, g_2). Following Step 3, we can establish a stability region for (g_1, g_2) in which the closed-loop system will remain stable. For example, in the special case when $g_1 = g_2$, the example system is guaranteed to be stable if $0 < g_1 = g_2 < 5.95$. That is, EUCON can maintain stability even if the execution time of every subtask becomes as high as 5.95 times its estimated one. Note this approach is also applicable to more complex systems following the same steps.

In our stability analysis, we assume the constrained optimization problem is *feasible*, i.e., there exists a set of task rates within their acceptable ranges that can make the utilization on every processor equal to its set point. If the problem is infeasible, no controller can guarantee the set point through rate adaptation. In this case, the system may switch to a different control adaptation mechanism (e.g., admission control or task reallocation). The integration of multiple adaptation mechanisms is part of our future work.

6.3. Control Tuning

For a system that is analytical stable, control tuning needs to consider the tradeoff between system oscillation and the speed of convergence. Severe oscillation in utilization is often undesirable in real world systems even if the average utilization remains close to the set point. In practice, this may lead to oscillation in application performance such as the frame rate of video and the frequency of control in process control systems. On the other hand, the speed of converge is important because it represents how quickly a system can recover from utilization variations and regain the desired utilization.

If the gains used in the controller (1 in the EUCON controller) is lower than the actual one (g_i), the real effect of the control input is going to be larger than what the controller has predicted and the utilization will oscillate around its set point. Therefore, using pessimistic estimation on execution times will reduce system oscillation because the system gains are less than 1 when execution times are overestimated. It should be noted that using pessimistic estimated execution times does *not* result in underutilization of the CPU. This key difference from open-loop scheduling is because EUCON dynamically adjusts rates based on *measured* utilization rather than the estimated execution times. On the other hand, more pessimis-

tic estimation on execution times leads to a smaller system gain, which can cause slower convergence to the set points.

7. Experimentation

7.1. Experimental Setup

We developed a simulation environment to simulate end-to-end tasks running on a distributed system. The simulation environment is composed of an event-driven simulator implemented in C++ and a controller implemented in MATLAB. The simulator implements the distributed real-time system controlled by EUCON, the utilization monitor and rate modulator. The subtasks on each processor are scheduled by the Rate Monotonic (RMS) scheduling algorithm [9]. The precedence constraints among subtasks are enforced by the release guard protocol [19]. The controller is based on the `lsqlin` least squares solver in MATLAB (version 6.0.0.88 release 12). The simulator opens a MATLAB process and initializes the controller at start time. In the end of each sampling period, the simulator collects the CPU utilization on each processor from the utilization monitors, and calls the controller in MATLAB with the utilization vector $\boldsymbol{u}(k)$ as parameters. The controller computes the control input, $\Delta\boldsymbol{r}(k)$, and return it to the simulator. The simulator then calls the rate modulator on each processor to adjust the task rates.

In all the experiments, we assume the each task's end-to-end deadline $d_i = n_i/r_i(k)$, where n_i is the number of subtasks in task T_i. We then evenly divide the deadline into the subdeadlines for its subtasks. The resultant sub-deadline of each subtask T_{ij} equals its period, $1/r_i(k)$. Hence we choose the schedulable utilization bound of RMS [9] as the utilization set point on each processor:

$$B_i = m_i(2^{1/m_i} - 1) \tag{13}$$

where m_i is the number of subtasks on P_i. All (sub)tasks meet their (sub)deadlines if the utilization set point on every processor is enforced. Other subdeadline assignment algorithms [7] and utilization bounds [8] may be applied to end-to-end scheduling. For example, when the deadlines of subtasks are different from their period, the schedulable utilization bound presented in [8] may be used as the utilization set points. Network delay is ignored in our simulations because it is not the focus of this paper. In practice, network delay can be handled by treating each network link as a processor [19], or considering the impact of worst-case network delay in subdeadline assignment.

Two different workload/system configurations were used in our experiments. The first configuration, SIMPLE, is the example used in the stability analysis in Section 6.2. The parameters of SIMPLE are listed in Table 1. The second column (*Proc*) represents the processor where a subtask is located. Subtasks T_{21} and T_{22} have the same rate parameters because they belong to a same task.

Table 1. Tasks parameters in SIMPLE

T_{ij}	*Proc*	c_{ij}	$1/R_{max,\,i}$	$1/R_{min,\,i}$	$1/r_i(0)$
T_{11}	P_1	35	35	700	60
T_{21}	P_1	35	35	700	90
T_{22}	P_2	35			
T_{31}	P_2	45	45	900	100

The second configuration, MEDIUM, simulates a medium-scale workload that is more typical in cluster-based DRE systems. MEDIUM includes 12 tasks (with a total of 25 subtasks) executing on 4 processors. To simulate real DRE systems, we adopt a mixed task model. There are 8 end-to-end tasks running on multiple processors and four local tasks (tasks T_8 to T_{12}). The execution time of every subtask in MEDIUM follows a uniform random distribution.

To evaluate the robustness of EUCON when execution times deviate from the estimated ones, the average execution time of each subtask T_{ij} can be changed by tuning a parameter called the *execution-time factor*, $etf_{ij}(k) = a_{ij}(k)/c_{ij}$, where $a_{ij}(k)$ is the average execution time of T_{ij}. The execution time factor represents how much the actual execution time of a subtask deviates from the estimated one. All subtasks share a same execution time factor in the presented experiments. In this case, the common execution time factor *etf* equals to the system gain on all processors in our system model, i.e., $etf = g_i$ ($1 \le i \le m$). The execution-time factor (and hence the average execution times) may be kept constant or changed dynamically in a run.

Two controllers are designed for SIMPLE and MEDIUM, respectively. Their parameters are listed in Table 2. The controller for MEDIUM has higher control and prediction horizons than that for SIMPLE because it needs to guarantee stability in a larger system.

Table 2. Controller parameters

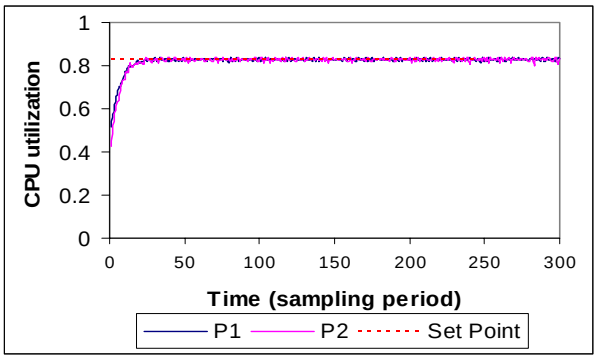
System	*P*	*M*	T_{ref}/T_s	T_s
SIMPLE	2	1	4	1000 time unit
MEDIUM	4	2		

In our experiments we consider the system performance as *acceptable* if the average utilization is within ±0.02 to the utilization set point, and the standard deviation is less than 0.05. The requirement on average utilization ensures the system achieves the desired utilization statistically. The requirement on standard deviation ensures that the utilization does not oscillate significantly. While the specific threshold for acceptable performance is dependent on specific applications, the general conclusions drawn in this section are applicable to other applications.

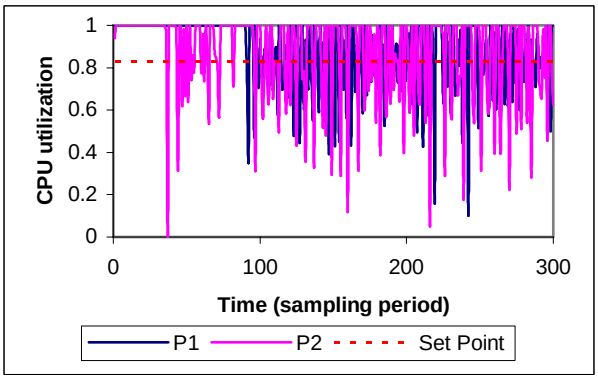
We compare EUCON against OPEN, an open-loop approach widely adopted in practice. Under OPEN, a designer assigns task rates based on *estimated* execution times so that $\boldsymbol{B} = \boldsymbol{Fr'}$ where $\boldsymbol{F}$ is the subtask allocation matrix defined in Section 5, and $\boldsymbol{r'}$ is the vector of task rates assigned by OPEN. OPEN results in desired utilization when estimated execution

times are accurate (i.e., when *etf* = 1). However, it causes underutilization of the CPU when execution times are overestimated estimated (i.e., when *etf* < 1), and over utilization of the CPU when execution times are overestimated (i.e., when *etf* > 1). This is a common problem because it is often difficult to establish tight bound on task execution times – especially on data-driven real-time systems whose execution times are heavily influenced by the value of sensor data or user input.

We present the results of two sets of simulation experiments. Experiment I evaluates system performance when task execution times deviate from the estimation. Experiment II stress-tests EUCON's ability to provide robust performance guarantees when task execution times varied dynamically at run-time.



(a) execution-time factor = 0.5



(b) execution-time factor = 7

Figure 3. CPU utilization under different execution time factors (SIMPLE)

7.2. Experiment I: Steady Execution Times

Since there are two subtasks on each of the processors in SIMPLE, the utilization set points $B_1 = B_2 = 0.828$ in Experiment I according to (13). All subtasks share a constant execution-time factor in each run, and different execution-time factors are used in different runs. Since the system gains g_1 and g_2 are equal to the execution-time factor under this setup, we can compare the results of our stability analysis to the simulation results through these experiments.

Figure 3(a) shows the system performance when the average execution time of every subtask is only *half* of the estimated one. In the beginning of the run, both processors are underutilized. EUCON then increases the task rates until the utilization of both processors converges to the utilization set points. In contrast, Figure 3(b) shows the situation when the average execution time of every subtask is *seven times* its estimation. In the beginning, the processors were fully utilized because of the long task execution times. At around time $30T_s$, the utilization drops sharply to almost zero and starts to oscillate. The utilization on P_2 also oscillates significantly. The system fails to converge to the utilization set point. This result is also consistent with our stability analysis that predicts the system will be unstable when the system gains exceed 5.95.

We plot the mean and standard deviation of utilization on P_1 during each run in Figure 4. Every data point is computed based on the measured utilization $u(k)$ from time $100T_s$ to $300T_s$ to exclude the transient response in the beginning of each run.

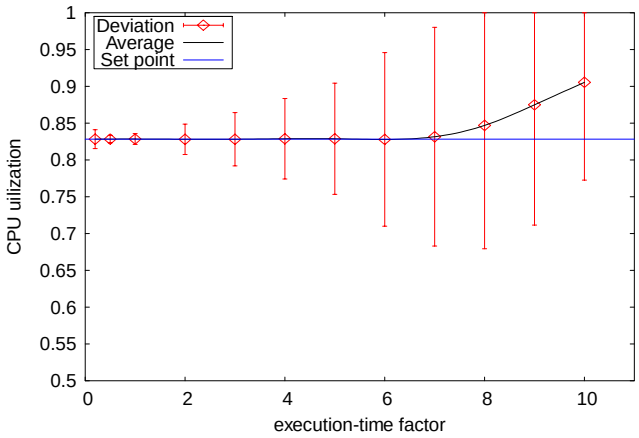


Figure 4. Average CPU utilization on Processor P_1 under different execution-time factors (SIMPLE)

In Figure 4, the average utilization remains close the utilization set point for execution-time factors between 0.2 and 6.5. However, it starts to deviate from the set point and increases linearly when the execution-time factor exceeds 6.5. Since stability in model predictive control is defined by the capability to converge to the set point, the experimental result indicates that the system clearly becomes unstable when the system gain is 7. This empirical result is close to our theoretical analysis that the system becomes unstable when the gain exceeds 5.95 (see Section 6). The standard deviation of utilization indicates the intensity of oscillation in a run. As the execution-time factor increases from 0.2 to 3, the standard deviation remains less than 0.05 and the average utilization remains within ±0.02 to the utilization set point. This result demonstrates that EUCON can enforce the same utilization guaran-

tees when execution times deviate from the estimated ones as long as the execution-time factor remains below 3. However, the standard deviation is higher than 0.05 for execution-time factors between 4 and 6, although the system is analytically stable in this range. This result is consistent with our analysis in Section 6 that pessimistic estimation on execution times will reduce system oscillation without underutilization the CPUs.

We then perform similar experiments under the MEDIUM configuration, which represents a typical medium-sized DRE system. Figure 5 plots the mean and standard deviation of utilization on processor P_1 in different runs while the execution-time factor is increased from 0.1 to 6.0 (the performance on other processors is similar and skipped due to space limit). For comparison, the expected utilization under OPEN is also plotted. OPEN causes underutilization when execution times are overestimated (i.e., *etf* < 1), and causes overload when execution times are underestimated (i.e., *etf* > 1). In contrast, EUCON provides acceptable utilization guarantees for any tested execution-time factor within the range [0.1, 1]. In this range, the average CPU utilization under EUCON remains within ±0.02 to the utilization set point and the standard deviation remains below 0.05. For example, when *etf* = 0.1, the average utilization under OPEN is only 0.073, while the average utilization under EUCON is 0.729 – the same as the utilization set point – with an standard deviation of 0.003. This result demonstrates EUCON provides desired utilization guarantees in a medium-sized DRE system even when actual execution times are significantly overestimated. Similar to SIMPLE, the oscillation of utilization in MEDIUM also increases as execution times are underestimated. This result re-confirms our observation that pessimistic estimation of execution times should be used in the predictive controller in EUCON.

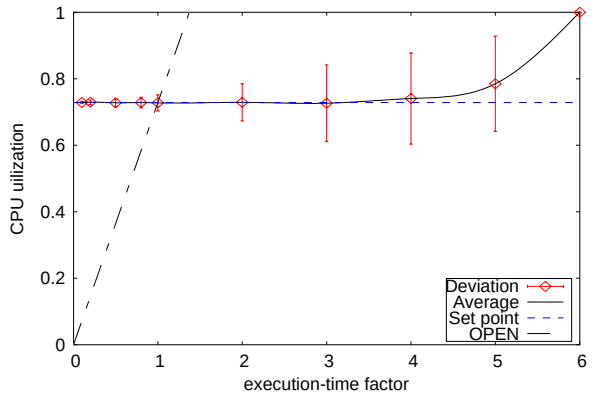


Figure 5. Average CPU utilization on P_1 under different execution-time factors (MEDIUM)

In summary, Experiment I shows that EUCON can maintain desired utilization guarantees on DRE systems when execution times are significantly overestimated. This property is particularly useful for DRE systems to avoid both CPU underutilization and overload when tight bounds on task execution times are not available, and for DRE middleware to achieve QoS portability when target platforms are not known *a priori*.

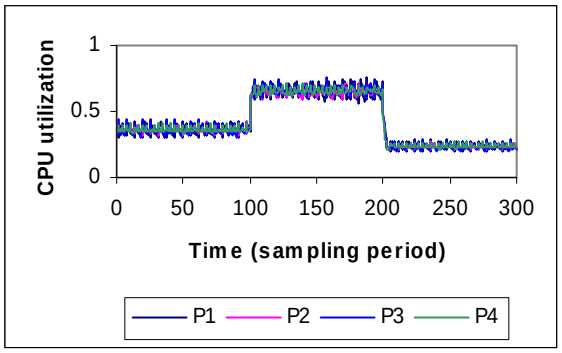


Figure 6. CPU Utilization under OPEN when execution times change dynamically

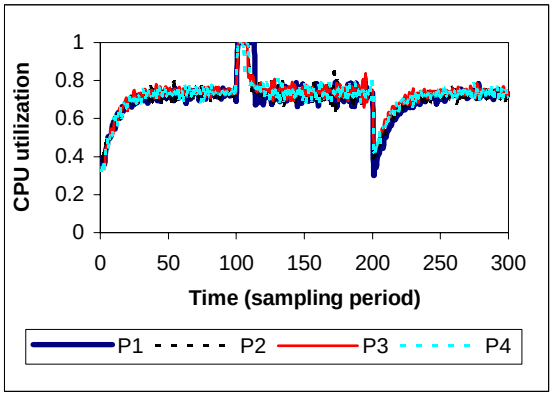


Figure 7. CPU Utilization under EUCON when execution times change dynamically

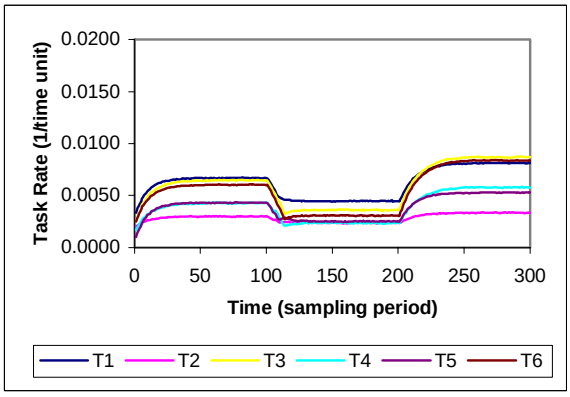


Figure 8. Task rates under EUCON when execution times change dynamically

7.3. Experiment II: Varying Execution Times

We then evaluate the system performance when execution times vary *dynamically* at run-time. In each run, the execution

time factor is initially 0.5. It is increased to 0.9 at time $100T_s$ to simulate an 80% increase in the execution times of all subtasks. At time $200T_s$, the execution-time factor is decreased to 0.33 to simulate a 67% decrease in execution times. Such instantaneous variation in workload stress tests the system capability of handling workload fluctuations [10].

As shown in Figure 6, the utilization under OPEN fluctuates significantly due to changes in execution times. This result is expected because in the OPEN approach task rates are not adjusted dynamically to compensate for the variations in system load. Clearly, traditional open-loop approaches such as OPEN cannot maintain desired utilization in face of varying execution times. In contrast, EUCON (as shown in Figure 7 and Figure 8) effectively maintains the utilization set points on all processors despite significant variations in execution times. At time $100T_s$, all processors are suddenly overloaded due to the instantaneous increase in execution times. EUCON responds to the system's deviation from the utilization set points by decreasing task rates. The utilization on all processors reconverges to their set points within $20T_s$. At time $200T_s$, the utilization dropped dramatically causing EUCON to increase task rates until the utilization on all processors re-converge to their set points. The system settling time in response to the utilization change at time $200T_s$ is longer than that at time $100T_s$. As discussed in Section 6 this is because the utilization gain is smaller during interval [$200T_s$, $300T_s$] than [$100T_s$, $200T_s$]. The system maintains stability and avoids significant oscillation throughout the run. Our experimental results demonstrate that EUCON can provide robust utilization guarantees when task execution times change dramatically at run time.

8. Conclusions

This paper extends the QoS control framework from single-processor to DRE systems. The challenging end-to-end utilization control problem is solved by a model predictive control approach. First, utilization control in a distributed system is formulated as a multi-variable constrained optimization problem. Second, a dynamic model is established to formally characterize the coupling among multiple processors due to end-to-end tasks and practical constraints. Third, a MIMO model predictive controller is designed to control the utilization on multiple processors simultaneously. Finally, stability analysis is performed to establish statistical guarantees on desired utilization despite the uncertainty introduced by variation in task execution times. Simulation results demonstrate that EUCON can provide robust utilization guarantees when task execution times are significantly overestimated and change dynamically at run-time. We are developing middleware services based on EUCON to provide QoS portability and real-time performance guarantees. In the future, we will develop decentralized control architecture to handle large-scale distributed systems.

References

[1] T.F. Abdelzaher, K.G. Shin, and N. Bhatti, "Performance Guarantees for Web Server End-Systems: A Control-Theoretical Approach," *IEEE Transactions on Parallel and Distributed Systems*, 13(1), Jan 2002.

[2] T.F. Abdelzaher, et. al., "Feedback Performance Control in Software Services," *IEEE Control Systems*, 23(3), June 2003.

[3] L. Abeni, L. Palopoli, G. Lipari, and J. Walpole, "Analysis of a Reservation-based Feedback Scheduler," *IEEE Real-Time Systems Symposium*, Dec 2002.

[4] M. Caccamo, G. Buttazzo, and L. Sha, "Handling Execution Overruns in Hard Real-Time Control Systems," *IEEE Transactions on Computers*, 51(7): 835-849, July 2002.

[5] J. Eker, *Flexible Embedded Control Systems-Design and Implementation*. Thesis, Lund Institute of Technology, Dec 1999.

[6] P.E. Gill, W. Murray, and M.H. Wright, *Practical Optimization*, Academic Press, London, UK, 1981.

[7] B. Kao and H. Garcia-Molina, "Deadline Assignment in a Distributed Soft Real-Time System," *IEEE Transactions on Parallel and Distributed Systems*, Dec. 1997.

[8] J.P. Lehoczky, "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines," *IEEE Real-Time Systems Symposium*, 1990.

[9] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of ACM*, 20(1): 46-61, 1973.

[10] C. Lu, J.A. Stankovic, G. Tao, and S.H. Son, "Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms," *Real-Time Systems Journal*, 23(1/2): 85-126, 2002.

[11] C. Lu, X. Wang, and C.D. Gill, "Feedback Control Real-Time Scheduling in ORB Middleware," *IEEE Real-Time and Embedded Technology and Applications Symposium*, May 2003.

[12] J.M. Maciejowski, *Predictive Control with Constraints,* Prentice Hall, 2002.

[13] Open Management Group, Real-Time CORBA Specification (Version 1.1), August 2002.

[14] D. Seto, et. al., "On Task Schedulability in Real-Time Control Systems," *IEEE Real-Time Systems Symposium*, Dec 1996.

[15] C. Shen, K. Ramamritham, and J.A. Stankovic, "Resource Reclaiming in Multiprocessor Real-Time Systems." *IEEE Transactions on Parallel and Distributed Systems* 4(4), 1993.

[16] L. Sha, R. Rajkumar, and J. Lehoczky, "Real-Time Synchronization Protocol for Multiprocessors," *IEEE Real-Time Systems Symposium*, 1988.

[17] J.A. Stankovic, et. al., "Feedback Control Real-Time Scheduling in Distributed Real-Time Systems," *IEEE Real-Time Systems Symposium*, 2001.

[18] D.C. Steere, et. al., "A Feedback-driven Proportion Allocator for Real-Rate Scheduling," *Symposium on Operating Systems Design and Implementation*, Feb 1999.

[19] J. Sun and J.W.S. Liu, "Synchronization Protocols in Distributed Real-Time Systems," *International Conference on Distributed Computing Systems*, 1996.