Report Number: WUCSE-2004-31

2004-05-01

Feedback Utilization Control in Distributed Real-Time Systems with End-to-End Tasks

Chenyang Lu, Xiaorui Wang, and Xenofon Koutsoukos

An increasing number of distributed real-time systems face the critical challenge of provid-ing quality of service guarantees in open and unpredictable environments. In particular, such systems often need to enforce utilization bounds on multiple processors in order to avoid over-load and meet end-to-end deadlines even when task execution times are unpredictable. While recent feedback control real-time scheduling algorithms have shown promise, they cannot han-dle the common end-to-end task model where each task is comprised of a chain of subtasks dis-tributed on multiple processors. This paper presents the End-to-end Utilization CONtrol (EU-CON) algorithm that adaptively maintains desired CPU utilization through... Read complete abstract on page 2.

Feedback Utilization Control in Distributed Real-Time Systems with End-to-End Tasks

Chenyang Lu, Xiaorui Wang, and Xenofon Koutsoukos

Complete Abstract:

An increasing number of distributed real-time systems face the critical challenge of provid-ing quality of service guarantees in open and unpredictable environments. In particular, such systems often need to enforce utilization bounds on multiple processors in order to avoid over-load and meet end-to-end deadlines even when task execution times are unpredictable. While recent feedback control real-time scheduling algorithms have shown promise, they cannot han-dle the common end-to-end task model where each task is comprised of a chain of subtasks dis-tributed on multiple processors. This paper presents the End-to-end Utilization CONtrol (EU-CON) algorithm that adaptively maintains desired CPU utilization through performance feed-backs loops. EUCON is based on a model predictive control approach that models utilization control 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 theo-retic analysis and simulations demonstrate that EUCON can provide robust utilization guaran-tees when task execution times deviate from estimation or vary significantly at run-time.

2004-31

Feedback Utilization Control in Distributed Real-Time Systems with End-to-End Tasks

Authors: Lu, Chenyang; Wang, Xiaorui; Koutsoukos, Xenofon

May 1, 2004

Abstract: An increasing number of distributed real-time systems face the critical challenge of providing quality of service guarantees in open and unpredictable environments. In particular, such systems often need to enforce utilization bounds on multiple processors in order to avoid overload and meet end-to-end deadlines even when task execution times are unpredictable. While recent feedback control real-time scheduling algorithms have shown promise, they cannot handle the common end-to-end task model 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 adaptively maintains desired CPU utilization through performance feedbacks loops. EUCON is based on a model predictive control approach that models utilization control 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 when task execution times deviate from estimation or vary significantly at run-time.

Feedback Utilization Control in Distributed Real-Time Systems with End-to-End Tasks[1]

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

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

Abstract

An increasing number of distributed real-time systems face the critical challenge of providing quality of service guarantees in open and unpredictable environments. In particular, such systems often need to enforce utilization bounds on multiple processors in order to avoid overload and meet end-to-end deadlines even when task execution times are unpredictable. While recent feedback control real-time scheduling algorithms have shown promise, they cannot handle the common end-to-end task model 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 adaptively maintains desired CPU utilization through performance feedbacks loops. EUCON is based on a model predictive control approach that models utilization control 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 when task execution times deviate from estimation or vary significantly at run-time.

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

[1] This is an extended version of a conference paper [15]. This work is funded, in part, by DARPA under grant NBCHC030140, and by NSF under grant CCR-0325529.

I. 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 on-line trading servers. A key challenge faced by such systems is providing critical quality of service (QoS) guarantees while the workload 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 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. Failure to meet critical QoS guarantees may result in loss of mission failures or severe financial damage.

These new systems require a paradigm shift from classical real-time computing that relies on accurate characterization of workloads and platform. Recently, control theoretic approaches that we call *QoS control* have shown promise in providing QoS guarantees in unpredictable environments. While classical 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 performance feedback loops. However, 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 [9][22]. 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 common in DRE systems, 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. First, 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. Second, 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 a same task. Changing the rate of a 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. Furthermore, QoS control is often subject to *constraints*. Examples include desired bounds on CPU utilization 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 maintain desired CPU utilization in distributed systems with end-to-end tasks in unpredictable environments through online adaptation. The primary contributions of this paper are three-fold: 1) derivation of a dynamic model that captures the coupling among processors and constraints in DRE systems executing end-to-end tasks; 2) development of a *Model Predictive Control* (MPC) approach for QoS control in DRE systems; and 3) design and control analysis of a distributed MIMO feedback control loop in EUCON that provide robust utilization guarantees when task execution times deviate from their estimation and vary significantly at run-time.

II. RELATED WORK

Traditional approaches for handling end-to-end tasks are based on the end-to-end scheduling [22] or distributed priority ceiling [19]. Both are open-loop approaches that rely on schedulability analysis that require *a priori* 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 [6][18]. A drawback of existing resource reclaiming techniques is that they often require 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.

A survey of feedback performance control in computing systems is presented in [2]. Recent research that applied control theory to real-time scheduling and utilization control is directly related to this paper. Steere et al., developed a feedback scheduler [21] that coordinated the CPU allocation to consumer and supplier threads. Abeni et al., presented control analysis of a reservation-based feedback scheduler [3]. Cervin et al. presented a feedback scheduler for digital control systems [7]. In [1], a feedback-based admission controller was designed to maintain desired utilization of an Apache web server. A Feedback Control real-time Scheduling (FCS) framework [13] was proposed to provide performance guarantees for real-time systems with unknown task execution times. The proposed FCS algorithms have been implemented as a middleware service [14]. All the aforementioned projects focused on controlling the performance of a *single* processor. In addition, their control designs are based on single-input-single-output linear control techniques. This control approach 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 [20]. However, FCS for distributed systems assumes, in contrast with the work presented in this paper, that tasks on different processors are *independent* from each other.

III. PROBLEM FORMULATION

We now formulate the utilization control problem in DRE systems.

A. 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 $T_{ij\text{-}1}$ is completed. We assume that a non-greedy synchronization protocol (e.g., release guard [22]) 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 [22]. Each task T_i is subject to an end-to-end relative deadline related to its period. In this work, we assume 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} known at design time. However, the *actual* execution time of a task may be significantly different from c_{ij} 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 task rates in many DRE applications (e.g., digital feedback control [7], sensor data update, and multimedia [5]) 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 utilization. Rate adjustment is an example of an adaptation mechanism that can be used to control utilization. Other adaptation mechanisms such as admission control and task reallocation may also be incorporated into the control framework.

EUCON is not designed to control network delays. Network delay may be handled by treating each network link as a processor [22], or by considering the impact of worst-case network delay in subdeadline assignment. EUCON can also be integrated with network-layer services such as IntServ [23] and DiffServ [4] to provide end-to-end delay guarantees.

B. Problem Formulation

Before formulating the utilization control problem, we introduce several notations.

- T_s: The sampling period.
- $u_i(k)$: The *CPU utilization* (or *utilization* for simplicity) of processor P_i in the kth 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 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 $(k+1)$th sampling period. 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 P_i. A higher weight w_i assigns higher importance to controlling $u_i(k)$.

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

$$\min_{\{r_j(k)|1\le j\le n\}} \sum_{i=1}^{n} w_i (B_i - u_i(k))^2 \text{ subject to two sets of constraints:}$$

$$u_i(k) \le B_i, \ (1 \le i \le n) \tag{1}$$
$$R_{min,i} \le r_i(k) \le R_{max,i}, \ (1 \le i \le m) \tag{2}$$

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.

C. Applications

EUCON has several important applications in a broad range of QoS-critical systems.

Meeting end-to-end deadlines: Real-time tasks must meet their end-to-end deadlines in DRE systems. In the end-to-end scheduling approach [22], the deadline of an end-to-end task is divided into subdeadlines of its subtasks, and the problem of meeting the deadline is transformed to the problem of meeting the subdeadline of each subtask. A well known approach for meeting the subdeadlines on a processor is by enforcing the *schedulable utilization bound* [12]. The subdeadlines of all the subtasks on a processor are guaranteed if the utilization of the processor remains below its schedulable utilization bound. To guarantee end-to-end deadlines, a user only needs to specify the utilization set point of each processor to a value below its schedulable utilization bound. EUCON can work with various subdeadline assignment algorithms [9][17] and schedulable utilization bounds for different task models [10][12] presented in the literature.

QoS portability: EUCON can also be deployed in a middleware to support *QoS portability* [14]. When an application is deployed on a faster platform, the task rates will be automatically increased to take advantage of the additional 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 utilization guarantees. EUCON's self-tuning capability can significantly reduce the cost of porting DRE software across platforms.

Overload protection: Many distributed systems (including non-real-time systems) must avoid saturation of processors, which may cause system crash or severe service degradation [1]. On COTS operating systems that support real-time priorities, high utilization by real-time threads may cause kernel starvation [14]. 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 additional 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 server clusters in which several processors connected through a high speed communication interface (e.g., a VME bus backplane). Many DRE systems (e.g., avionics systems, shipboard computing, and process control systems) fall into this category. A centralized QoS control architecture is usually sufficient to this class of DRE systems. Decentralized control for large-scale systems is part of our future work.

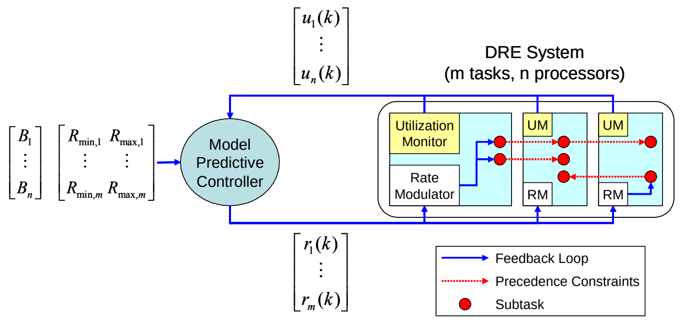

$\begin{bmatrix} u_1(k) \\ \vdots \\ u_n(k) \end{bmatrix}$
DRE System
(m tasks, n processors)
$\begin{bmatrix} B_1 \\ \vdots \\ B_n \end{bmatrix}$
$\begin{bmatrix} R_{\min,1} & R_{\max,1} \\ \vdots & \vdots \\ R_{\min,m} & R_{\max,m} \end{bmatrix}$
Model Predictive Controller
Utilization Monitor
Rate Modulator
UM
UM
RM
RM
$\begin{bmatrix} r_1(k) \\ \vdots \\ r_m(k) \end{bmatrix}$
Feedback Loop
Precedence Constraints
Subtask

Figure 1. The MIMO feedback control loop in EUCON

IV. OVERVIEW OF EUCON

EUCON 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 [14]) 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 new task rates $\boldsymbol{r}(k) = \boldsymbol{r}(k\text{-}1) + \Delta\boldsymbol{r}(k)$ to the rate modulator on each processor through its feedback lane.
3. The rate modulator on each processor changes the task rates according to $\boldsymbol{r}(k)$.

Since the core of EUCON is the controller, we will focus on its design in the rest of the paper. The design of the other components is similar to FCS/nORB [14], a feedback control scheduling service on an Object Request Broker middleware.

V. 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. 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 subtask T_{jl} (the l^{th} subtask 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

$$u(k) = \begin{bmatrix} u_1(k) \\ u_2(k) \end{bmatrix},\ G = \begin{bmatrix} g_1 & 0 \\ 0 & g_2 \end{bmatrix},\ \Delta b(k) = \begin{bmatrix} \Delta b_1(k) \\ \Delta b_2(k) \end{bmatrix},\ F = \begin{bmatrix} c_{11} & c_{21} & 0 \\ 0 & c_{22} & c_{31} \end{bmatrix},\ \Delta r(k) = \begin{bmatrix} \Delta r_1(k) \\ \Delta r_2(k) \\ \Delta r_3(k) \end{bmatrix}$$

From (5-6), the system model is

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

VI. Design and Analysis of a Model Predictive Controller

We present the design and analysis of a model predictive controller for EUCON. We first derive a mathematical formulation of EUCON in the model predictive control framework. Next this formulation is transformed to a constrained least-squares problem, which allows us to design the control algorithm based on an existing least squares solver. Finally, we prove the stability of our controller through control analysis.

A. A Formulation for Model Predictive Control

Based on the system model, a MIMO predictive controller can be designed to guarantee the utilization set points on multiple processors. The single-input-single-output (SISO), linear control approach adopted in earlier works on feedback control real-time scheduling [13][20] 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)* [16] approach. MPC is an advanced control technique used extensively in industrial process control. 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 where the performance measures and the coupling between processors can be expressed by constraints and MIMO system models. 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*. The notation $\boldsymbol{x}(k+i|k)$ means that the vector signal $\boldsymbol{x}$ depends on the conditions at time *k*. 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 next step, the prediction horizon slides one sampling period and the input is computed again as a solution to a constrained optimization problem based on performance feedbacks ($\boldsymbol{u}(k)$). MPC combines performance prediction, optimization, constraint satisfaction, and feedback control into a single algorithm. Details of MPC can be found in [16].

We now design a controller for EUCON. The 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. The cost function to be minimized by our controller is

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

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 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 behaves like a linear system.

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

T_{ref} is the time constant that specifies the speed of system response. A larger T_{ref} causes the system to converge faster to the set point. By minimizing the tracking error, the closed-loop system will converge to the utilization set point if the system is stable. The weight matrix $\boldsymbol{Q}(i)$ can be tuned to represent preferences between processors. For example, a higher weight may be assigned 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 IV. However, the model cannot be directly used by the controller because the system gains $\boldsymbol{G}$ are unknown. Therefore, the controller needs to use an approximate model. Our controller assumes $\boldsymbol{G} = [1 \ldots 1]^T$ 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

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

Although this approximate model may behave differently from the real system, as we prove in Section V.C, 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 of the closed-loop system. 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 transformed to a standard constrained *least-squares* problem [16]. The controller can then use a standard *least-squares* solver to solve this problem on-line. In the following subsection we present this transformation.

B. Transformation to Least-Squares Problem

A standard constrained least-squares problem is in the form of

$$\min_{s(k)}\left(\sum_{i=1}^{P}\left\|\Theta s(k)-E(k)\right\|_{Q(i)}^{2}+\sum_{i=0}^{M-1}s(k)_{R(i)}^{2}\right) \text{ subject to constraints } \mathbf{\Omega s}(k)\le \boldsymbol{\omega} \tag{10}$$

s(*k*) denotes the vector of change to the control input in the control horizon. In EUCON,

$$s(k)=\begin{bmatrix}\Delta r(k\,|\,k)-\Delta r(k-1)\\ \vdots \\ \Delta r(k+M-1\,|\,k)-\Delta r(k+M-2\,|\,k)\end{bmatrix}$$

To transform our control problem to a least-squares problem, we re-write our cost function in (7) and constraints (1-2) in the form (10). Since the control penalty terms in (7) is consistent with (10), we only need to transform the tracking error term in (7) and the constraints (1-2) to formulations in terms of *s*(*k*). First we work on the tracking error term in (7). From the plant model (6) and (9), the predicted utilization for given prediction horizon can be written as:

$$\begin{bmatrix}u(k+1\,|\,k)\\ \vdots \\ u(k+M\,|\,k)\\ u(k+M+1\,|\,k)\\ \vdots \\ u(k+P\,|\,k)\end{bmatrix}=\begin{bmatrix}u(k)\\ \vdots \\ u(k)\\ u(k)\\ \vdots \\ u(k)\end{bmatrix}+\begin{bmatrix}F\\ \vdots \\ \sum_{i=0}^{M-1}F\\ \sum_{i=0}^{M}F\\ \vdots \\ \sum_{i=0}^{P}F\end{bmatrix}\Delta r(k-1)+\begin{bmatrix}F & 0 & \cdots & 0\\ F+F & F & \cdots & 0\\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=0}^{M-1}F & \sum_{i=0}^{M-2}F & \cdots & F\\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=0}^{P-1}F & \sum_{i=0}^{P-2}F & \cdots & \sum_{i=0}^{P-M}F\end{bmatrix}s(k) \tag{11}$$

We can rewrite (11) as:

$$u'(k)=u(k)+\Gamma\Delta r(k-1)+\Theta s(k) \tag{12}$$

$$u'(k)=\begin{bmatrix}u(k+1\,|\,k)\\ \vdots \\ u(k+M\,|\,k)\\ u(k+M+1\,|\,k)\\ \vdots \\ u(k+P\,|\,k)\end{bmatrix},\quad \Gamma=\begin{bmatrix}F\\ \vdots \\ \sum_{i=0}^{M-1}F\\ \sum_{i=0}^{M}F\\ \vdots \\ \sum_{i=0}^{P}F\end{bmatrix},\quad \Theta=\begin{bmatrix}F & 0 & \cdots & 0\\ F+F & F & \cdots & 0\\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=0}^{M-1}F & \sum_{i=0}^{M-2}F & \cdots & F\\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=0}^{P-1}F & \sum_{i=0}^{P-2}F & \cdots & \sum_{i=0}^{P-M}F\end{bmatrix} \tag{13}$$

In addition, we define

$$\boldsymbol{E}(k) = \boldsymbol{ref'}(k) - \boldsymbol{u}(k) - \Gamma\Delta\boldsymbol{r}(k\text{-}1) \tag{14}$$

where ***ref'***(*k*) represents the reference trajectory for specified prediction horizon:

$$ref'(k) = \begin{bmatrix} ref(k+1\,|\,k) \\ \vdots \\ ref(k+P\,|\,k) \end{bmatrix}$$

Given Θ and ***E***(*k*) in (12) and (13), our cost function (7) is equivalent to the one in the least-squares problem (10). We now transform the constraints (1-2) to the linear inequality constraint form as $\boldsymbol{\Omega s}(k) \le \boldsymbol{\omega}$. Firstly the rate constraint (1) in control horizon *M* as:

$$\begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \\ -1 & 0 & \cdots & 0 \\ 0 & -1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & -1 \end{bmatrix} \begin{bmatrix} r(k) \\ r(k+1) \\ \vdots \\ r(k+M-1) \end{bmatrix} \le \begin{bmatrix} R_{\max} \\ R_{\max} \\ \vdots \\ R_{\max} \\ -R_{\min} \\ -R_{\min} \\ \vdots \\ -R_{\min} \end{bmatrix}$$

From ***r***(*k*) = ***r***(*k*-1) + Δ***r***(*k*), the above inequality is equivalent to

$$\begin{bmatrix} 1 & 0 & \cdots & 0 \\ 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \\ -1 & 0 & \cdots & 0 \\ -1 & -1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ -1 & -1 & \cdots & -1 \end{bmatrix} \begin{bmatrix} \Delta r(k) \\ \Delta r(k+1) \\ \vdots \\ \Delta r(k+M-1) \end{bmatrix} \le -\begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ -1 \\ -1 \\ \vdots \\ -1 \end{bmatrix} r(k-1) + \begin{bmatrix} R_{\max} \\ R_{\max} \\ \vdots \\ R_{\max} \\ -R_{\min} \\ -R_{\min} \\ \vdots \\ -R_{\min} \end{bmatrix}$$

From Δ***r***(*k*) = Δ***r***(*k*-1) + (Δ***r***(*k*)- Δ***r***(*k*-1)), we can transform the rate constraints to the following linear inequality constraints:

$$\begin{bmatrix} 1 & 0 & \cdots & 0 \\ 2 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ M & M-1 & \cdots & 1 \\ -1 & 0 & \cdots & 0 \\ -2 & -1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ -M & -(M-1) & \cdots & -1 \end{bmatrix} s(k) \le -\begin{bmatrix} 1 \\ 2 \\ \vdots \\ M \\ -1 \\ -2 \\ \vdots \\ -M \end{bmatrix} \Delta r(k-1) - \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ -1 \\ -1 \\ \vdots \\ -1 \end{bmatrix} r(k-1) + \begin{bmatrix} R_{\max} \\ R_{\max} \\ \vdots \\ R_{\max} \\ -R_{\min} \\ -R_{\min} \\ \vdots \\ -R_{\min} \end{bmatrix} \tag{15}$$

Now we consider the utilization bound constraints (1). From (1) and (12) the utilization bound constraints are equivalent to the following linear inequality

$$\Theta \boldsymbol{s}(k) \le -\boldsymbol{u}(k) - \Gamma \Delta \boldsymbol{r}(k-1) + \boldsymbol{B} \tag{16}$$

We have transformed our MPC formulation to a constrained least-square formulation described by (10, 12-15). Since the constraints (15-16) depend on $\boldsymbol{u}(k)$, $\Delta\boldsymbol{r}(k\text{-}1)$, and $\boldsymbol{r}(k\text{-}1)$, both of them are known at time k. We can use any standard *least-squares* solver to solve this control problem now. In our simulator, we implement the controller based on the `lsqlin` solver in Matlab, which uses an active set method similar to that described in [8]. 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. While our controller is capable of handling medium-scale systems which are the focus of this paper, more efficient control algorithm may be needed by large systems. A preliminary overhead measurement in the MATLAB environment is presented in Section VII.F.

C. Stability Analysis

A dynamic system is *stable* iff for every initial condition it will converge to the equilibrium point [16]. In our case, the equilibrium points of the system are the utilization set points $\boldsymbol{B}$. Hence a stable DRE system guarantees that the utilization on every processor converge to its set point. We first outline a general approach for analyzing the stability for a DRE system controlled by EUCON and then give an example.

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

   $$\boldsymbol{u}(k+1) = \boldsymbol{A}\boldsymbol{u}(k) + \boldsymbol{C} \tag{17}$$

   where $\boldsymbol{A}$ is a matrix whose eigenvalues depend on the system gains $\{g_i \mid 1 \le i \le n\}$.
3. Derive the stability condition of the closed-loop system described by (11). 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.

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. The model-predictive control formulation facilitates this integrated solution because the infeasibility of an adaptation mechanism can be detected by least-square solver and, in turn, triggers a new adaptation mechanism.

Example: We now apply the stability analysis approach to the example system described in the end of Section V. 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

$$V(k) = \sum_{j=1}^{2}\sum_{i=1}^{2}(u_j(k+i\,|\,k) - ref_j(k+i\,|\,k))^2 + \sum_{j=1}^{3}(\Delta r_j(k) - \Delta r_j(k-1))^2 \quad (18)$$

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

$$\begin{bmatrix} u_1(k+1) \\ u_2(k+1) \\ u_1(k+2) \\ u_2(k+2) \end{bmatrix} = \begin{bmatrix} u_1(k) \\ u_2(k) \\ u_1(k+1) \\ u_2(k+1) \end{bmatrix} + \begin{bmatrix} c_{11} & c_{21} & 0 \\ 0 & c_{22} & c_{31} \\ c_{11} & c_{21} & 0 \\ 0 & c_{22} & c_{31} \end{bmatrix} \begin{bmatrix} \Delta r_1(k) \\ \Delta r_2(k) \\ \Delta r_3(k) \end{bmatrix} \quad (19)$$

After we substitute (19) and the reference trajectory (7) to (18), the cost function becomes a function of $\mathbf{\Delta r}(k)$. We then derive the control input vector $\mathbf{\Delta r}(k)$ that minimize the cost function through partial differentiation. Following Step 2, we establish the closed-loop model by substituting $\mathbf{\Delta r}(k)$ 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 derive the conditions in terms of (g_1, g_2) under 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 close to 5.95 times its estimated one. The details of the stability analysis on this example are not shown here due to the page limit.

D. Control Tuning

For a stable system, controller tuning involves a tradeoff between utilization oscillation and the speed of convergence. Severe oscillation in utilization is undesirable even if the average utilization remains close to the set point. In practice, this may lead to oscillation in application performance such as video frame rate and the frequency of control in process control systems. The speed of converge is also 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 EUCON) 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 system will oscillate. 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 under EUCON does *not* cause underutilization. This key difference from open-loop scheduling is because EUCON dynamically adjusts rates based on *measured* utilization rather than the estimated execution times. However, more pessimistic estimation on execution times leads to smaller gains, which cause slower convergence to the set points.

The choice of the sampling period must balance convergence time, overhead, and oscillation. A short sampling period speeds up convergence by enabling the system to adapt to variations at a higher frequency. However, a short sampling period also increases the run-time overhead of EUCON because its feedback control loop is invoked once per sampling period. Moreover, since EUCON measures the *average* utilization over a sampling period, a longer sampling period may filter out noise in the utilization input to the controller and hence reduce oscillation.

VII. EXPERIMENTATION

A. Experimental Setup

Our simulation environment is composed of an event-driven simulator implemented in C++ and a controller implemented in MATLAB (R12). 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 [12]. The precedence constraints among subtasks are enforced by the release guard protocol [22]. The controller is based on the `lsqlin` least squares solver in MATLAB. 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.

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. Each end-to-end deadline is evenly divided into subdeadlines for its subtasks. The resultant subdeadline of each subtask T_{ij} equals its period, $1/r_i(k)$. Hence the schedulable utilization bound of RMS [12] is used as the utilization set point on each processor:

$$B_i = m_i(2^{1/m_i} - 1) \tag{20}$$

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. As discussed in Section III.C, other subdeadline assignment algorithms [9] and utilization bounds [10] may also be used with EUCON. Network delay is ignored in the simulations.

Table 1: Task parameters in SIMPLE (*Proc* represents the processor where a subtask is located)

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

Two different workload/system configurations were used in our experiments. SIMPLE (see Table 1) is the example used in the stability analysis in Section VI.C. The second configuration, MEDIUM, simulates a more complex workload. MEDIUM includes 12 tasks (with a total of 25 subtasks) executing on 4 processors. There are eight end-to-end tasks running on multiple processors and four local tasks (tasks T_8 to T_{12}). The execution time of every subtask T_{ij} in MEDIUM follows a uniform distribution. The parameters of MEDIUM are available in Appendix B.

Table 2: Controller parameters

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

To evaluate the robustness of EUCON when execution times deviate from the estimation, 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} 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. The execution-time factor (and hence the average execution times) may be kept constant or changed dynamically in a run. When all subtasks share a same constant execution time factor etf, etf equals to the system gain on every processor in the model, i.e., $etf = g_i$ ($1 \le i \le m$). The controller parameters are listed in Table 2. The controller for MEDIUM has higher control and prediction horizons to guarantee stability in a larger system.

B. Baselines

We compare EUCON against two baseline algorithms, OPEN and FC-U-E2E. OPEN is an open-loop algorithm that uses fixed task rates. It assigns task rates *a priori* based on *estimated* execution times so that $\boldsymbol{B} = \boldsymbol{F}\boldsymbol{r}'$, where $\boldsymbol{F}$ is the subtask allocation matrix defined in Section IV, and $\boldsymbol{r}'$ is the vector of task rates assigned by OPEN. From the definition of $etf(k)$ we have

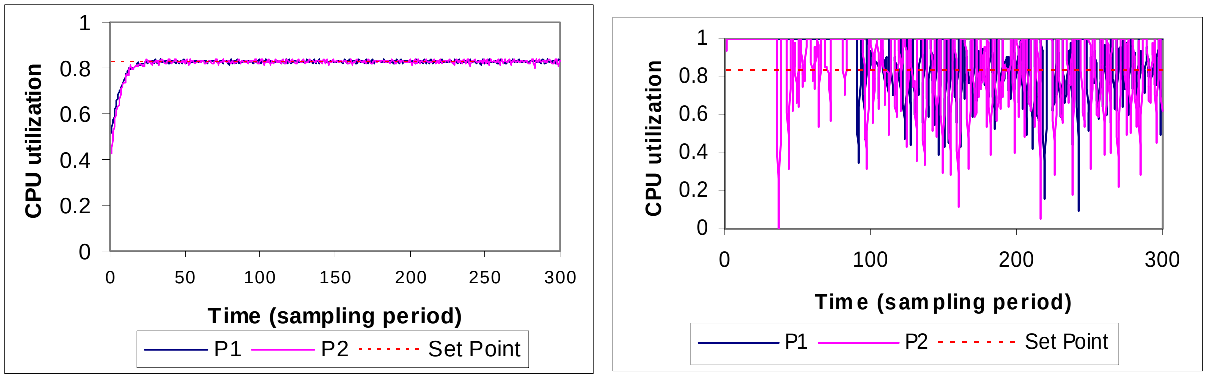
$$\boldsymbol{u}(k) = etf(k)\boldsymbol{B} \tag{21}$$

Although OPEN can result in desired utilization when estimated execution times are accurate (i.e., $etf(k) = 1$), it causes underutilization when execution times are overestimated (i.e., $etf(k) < 1$), and CPU over-utilization when execution times are underestimated (i.e., $etf(k) > 1$).

Unfortunately, it is often difficult to establish tight bound on task execution times – especially in open and unpredictable environments where task execution times are heavily influenced by the value of sensor data or user input at run time.

FC-U-E2E is an extension of the FC-U [13] algorithm. Similarly to EUCON, FC-U features a feedback control loop that controls utilization by dynamically adjusting task rates. However, FC-U is a *single-processor* algorithm, i.e., it only controls the utilization of a *single* processor. It uses a SISO Proportional controller to compute the changes to task rates based on measured utilization. A simple approach for utilization control in a distributed system is executing a FC-U algorithm on each processor. Each FC-U algorithm controls the utilization of its own processor by computing task rates *independently* from others. However, this approach cannot handle the end-to-end task model due to its constraint that all the subtasks of an end-to-end task must execute at the same rate. In contrast, FC-U algorithms on those processors may decide to assign different rates to the same task based on the states of their own processors. For example, the FC-U controller on a heavily loaded processor may assign a lower rate to a task than that assigned by a lightly loaded processor that shares the same task. Therefore conflicts among the desired rates by multiple processors must be resolved. To guarantee the utilization bound constraints on all processors, a conservative approach can be adopted to assign the *lowest* rate given by any processors to a task. This mechanism can be implemented by adding a *min* component to the rate modulator on each processor. In the end of every sampling period, the rate modulator on each processor P_i receives the rates assigned to each of its tasks from all the FC-U controllers on processors that share tasks with P_i, and change the rate of each of its task to the minimum one among all the received rates for this task. We refer to this extended algorithm *FC-U-E2E*. A fundamental difference between EUCON and FC-U-E2E is that EUCON *explicitly* incorporates the inter-processor coupling in a distributed system in its the design of a MIMO MPC, while FC-U-E2E *implicitly* handles the coupling by resolving the conflict among multiple SISO Proportional controllers through a `min` operator. As a baseline FC-U-E2E allows us to study the benefit of MPC compared to simple linear control.

In the following, we present three sets of simulations. In Experiment I, execution times are steady but deviate from the estimation. In Experiment II, task execution times vary dynamically at run-time. Experiment III compares EUCON with FC-U-E2E.



(a) execution-time factor = 0.5 (b) execution-time factor = 7

Figure 2: Utilization under different execution time factors (SIMPLE)

C. Experiment I: Steady Execution Times

In this set of experiments, all subtasks share a constant execution-time factor in each run. Since the system gains g_1 and g_2 equal the execution-time factor under this setup, we can compare the results of our stability analysis to the simulation results through these experiments. Figure 2(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. As predicted by our control analysis, the system remains stable in this case. In contrast, Figure 2(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.

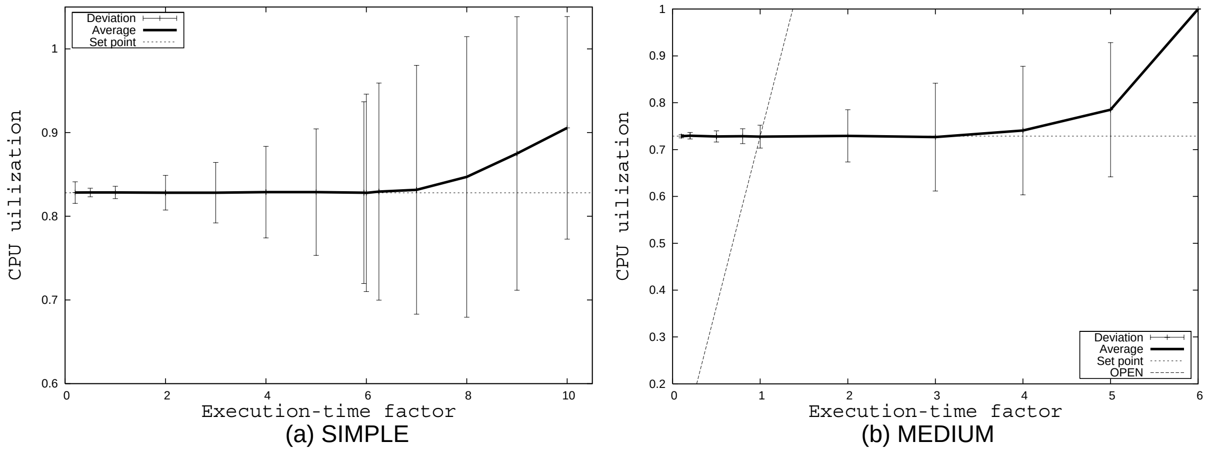


Figure 3: Average utilization on P_1

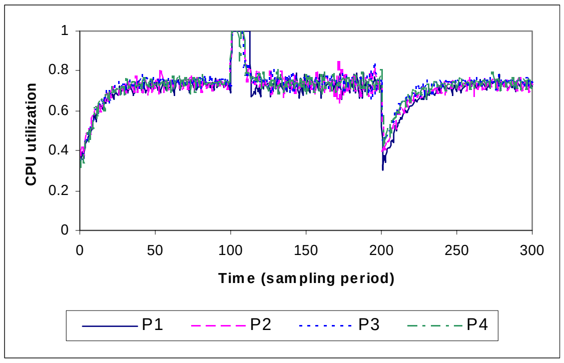
We plot the mean and standard deviation of utilization on P_1 during each run in Figure 3(a). Every data point is based on the measured utilization $u(k)$ from time $100T_s$ to $300T_s$ to exclude the transient response in the beginning of each run. The system performance is considered *acceptable* if the average utilization is within ±0.02 to the utilization set point, and the standard deviation is less than 0.05. Satisfying the requirement on average utilization ensures that the

Courier, 24

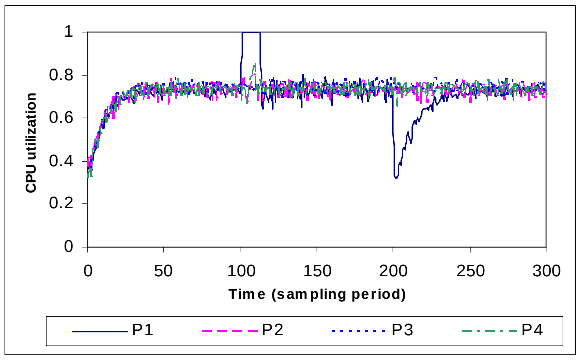
system achieves the desired utilization. Satisfying the requirement on standard deviation ensures that the utilization does not oscillate significantly. While the thresholds for acceptable performance depend on specific applications, the general conclusions drawn in this section are applicable to many applications. As shown in Figure 3(a), the average utilization remains close to the set point for execution-time factors between 0.20 and 5.95, and it starts deviating from the set point and increases linearly when the execution-time factor exceeds 6.00. When execution-time factor = 5.95, the average utilizations on P_1 and P_2 are 0.828 and 0.829, respectively. When execution-time factor increases to 6.00, however, the average utilization on P_1 and P_2 become 0.828 and *0.833*, respectively. Based on the set point of 0.828 on both processors, the system becomes unstable (on P_2) when execution-time factor is in the range [5.95, 6.00] in the run. This empirical result is close to the analysis which shows the system should remain stable when the gain is below 5.95 (see Section V).

The standard deviation of utilization indicates the intensity of oscillation. 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 set point. These results demonstrate that EUCON can enforce the *same* utilization guarantees when execution times deviate from the estimates 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 V that pessimistic estimation on execution times will reduce oscillation without underutilizing the CPUs.

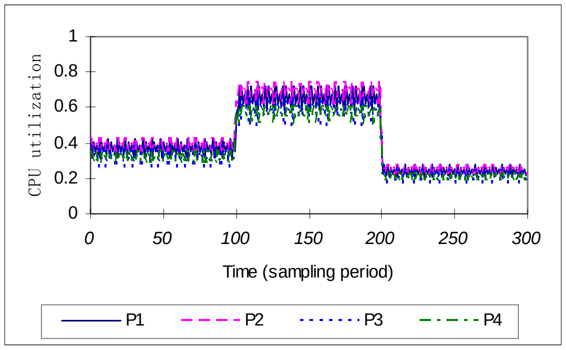
We then repeat our experiments under MEDIUM in order to evaluate the system performance under more complex settings. Figure 3(b) plots the mean and standard deviation of utilization on processor P_1 under different execution-time factors (the performance on other processors is similar to P_1 and is not shown due space limit). For comparison, the expected utilization under OPEN (computed based on (21)) is also plotted. OPEN causes underutilization when execution times are overestimated (*etf* < 1), and causes overload when execution times are underestimated (*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 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 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 can achieve desired utilization even when execution times are significantly overestimated. Similar to SIMPLE, the oscillation of utilization under 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.



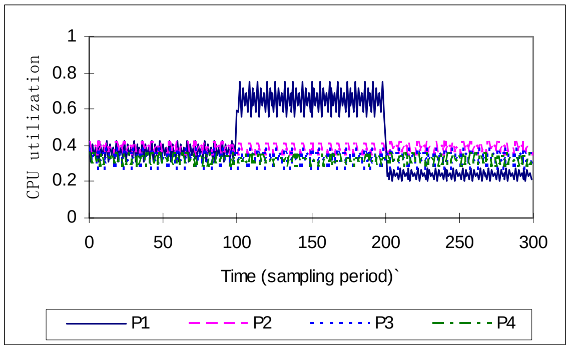
(a) Global fluctuation (EUCON)



(b) Local fluctuation on P_1 (EUCON)



(c) Global fluctuation (OPEN)



(d) Local fluctuation on P_1 (OPEN)

Figure 4 Utilization and task rates when execution times fluctuate at run time

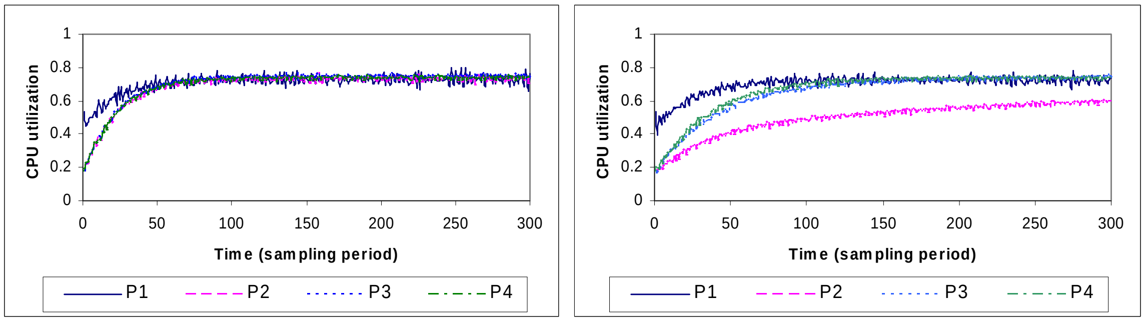
D. Experiment II: Varying Execution Times

In Experiment II, execution times vary *dynamically* at run-time under the MEDIUM configuration. To investigate the robustness of EUCON we tested two scenarios of workload fluctuation. In the first set of runs, the average execution times on all processors change uniformly.

In the second set of runs, only the average execution times on P_1 change dynamically. The first scenario represents *global* load fluctuation in the whole system, while the second scenario represents *local* fluctuation on a part of the system.

In each run with global workload fluctuation, the execution time factor is initially 0.5. At time $100T_s$, it increases to 0.9 causing an 80% increase in the execution times of all subtasks. At time $200T_s$, the execution-time factor drops to 0.33 causing a 67% decrease in execution times. Such instantaneous variation in workload stress tests the system capability of handling workload fluctuations [13]. As shown in Figure 4(a), EUCON enforces 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 increase in execution times. EUCON responds to the deviation from the utilization set points by decreasing task rates. The utilization on all processors re-converges 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 regain to their set points. The system settling time after $200T_s$ is longer than that follows $100T_s$. As discussed in Section V this is because the system 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 despite variations in execution times. In contrast, Figure 4(c) shows that the utilization under OPEN fluctuates significantly because it cannot adapt to the workload variations.

In each run with local workload fluctuation, the execution-time factor on P_1 follows the same variation as that in global fluctuation, but all the other processors have a fixed execution-time factor of 0.5. As shown in Figure 4(b), the utilization of P_1 converges to its set point after the significant variation of execution times at $120T_s$ and $250T_s$, respectively. The settling times under local workload fluctuation are close to those under global workload fluctuation. We also observe that the other processors experience only slight utilization fluctuation after the execution times change on P_1. This result demonstrates that EUCON effectively handles the coupling among processors during rate adaptation. In contrast, OPEN fails to maintain steady utilization on P_1 in face of local workload fluctuation (as shown in Figure 4(d)).



(a) EUCON (b) FC-U-E2E

Figure 5 Utilization under EUCON and FC-U-E2E (etf = 0.2, MEDIUM)

E. Experiment III: Comparison with FC-U-E2E

A premise of this work is that the MIMO approach adopted by EUCON can outperform the SISO control approach. SISO control cannot handle the coupling among processors effectively - especially when the utilization on different processors are unbalanced. In this situation, the task rates computed by different controllers may become inconsistent with each other due to the unbalanced utilization on different processors. We now compare the performance of EUCON and FC-U-E2E under an unbalanced workload. The workload used in this experiment is the same as MEDIUM except that the execution times on processor P_1 are higher. The execution time factor remains at 0.2 in each run. As shown in Figure 5(a), the utilization on all processors converge to their set points despite the difference in initial values when EUCON is used. The performance of FC-U-E2E is shown in Figure 5(b). The utilization on P_1 follows a similar trajectory as under EUCON. However, all the other three processors suffer from significantly longer settling times. For instance, while it only takes about $60T_s$ for P_4 to reach its set point under EUCON, it fails to reach its set point in the end of the run ($300T_s$). Long under EUCON, it fails to reach its set point in the end of the run ($300T_s$). Long settling times are undesirable because systems need to quickly recover from load variation.

We now analyze what causes the poor performance of FC-U-E2E. After P_1 reaches the set point at time $50T_s$, its Proportional controller stops increasing the rates of all tasks with subtasks on this processor. Because all tasks must execute at the *lowest* rate given by any controllers in FC-U-E2E, their rates will stop increasing, even if the controllers on the other processors need them to do so in order to reach their set points. This effectively slows down the convergence of processors $P_{2\text{-}4}$ to their set points. Actually, FC-U-E2E can eventually reach the set points only because every processor has a local task whose rate can be changed independently from other processors. P_2 has the longest settling time because it shares four end-to-end tasks P_1, while each of P_3 and P_4 only shares two with P_1. Hence the utilization of P_2 is particularly affected by the controller on P_1. After P_3 and P_4 both reach their set points, the utilization increase of P_2 becomes even slower since only its local task can increase its rate in this case. Compared with FC-U-E2E, a key advantage of EUCON lies in its capability to handle the coupling among multiple processors. Furthermore, MPC provides a theoretic framework to analyze system stability under a wide range of execution-time factors.

F. Overhead

To estimate the run-time overhead of the controller, we measure the execution time of the least squares solver which dominates the computation cost of the controller. In the simulations with the MEDIUM configuration on a 1.99GHz Pentium 4 PC with 256MB RAM, each invocation of the solver in MATLAB takes less than 9ms (corresponding to less than 1% CPU utilization when the sampling period is 1 sec). This result indicates the overhead of the controller is acceptable for a range of applications. Since this preliminary result is based on the solver in the MATLAB environment, it is not a precise benchmark for a controller implemented in native code. Evaluation of EUCON in a real middleware environment is part of our future work.

VIII. CONCLUSIONS

EUCON features a model predictive controller to handle the coupling among multiple processors and constraints based a mathematical model that characterizes the dynamics of distributed systems with end-to-end tasks. Both stability analysis and simulation results demonstrate that EUCON can maintain desired utilization on multiple processors when task execution times are significantly overestimated and change dynamically at run-time. EUCON also outperforms both open-loop scheduling and a FCS algorithm based on SISO linear control.

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, J.A. Stankovic, C. Lu, R. Zhang, and Y. Lu, "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," RTSS, 2002.

[4] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services," IETF RFC 2475, 1998.

[5] S. Brandt and G.J. Nutt, "Flexible Soft Real-Time Processing in Middleware," Real-Time Systems 22(1-2), 2002.

[6] 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.

[7] A. Cervin, J. Eker, B. Bernhardsson, and K.-E. Årzén, "Feedback-Feedforward Scheduling of LQG-Control Tasks," Real-time Systems Journal, 23, 2002.

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

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

[10] J.P. Lehoczky, "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines," RTSS, 1990.

[11] B. Li and K. Nahrstedt, "A Control-based Middleware Framework for QoS Adaptations," IEEE JSAC, 17, Sept. 1999.

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

[13] 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.

[14] C. Lu, X. Wang and C.D. Gill, "Feedback Control Real-Time Scheduling in ORB Middleware," RTAS, 2003.

[15] C. Lu, X. Wang, and X. Koutsoukos, "End-to-End Utilization Control in Distributed Real-Time Systems," ICDCS, 2004.

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

[17] M.D. Natale and J.A. Stankovic, "Dynamic End-to-End Guarantees in Distributed Real Time Systems," RTSS, 1994.

[18] 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.

[19] L. Sha, R. Rajkumar and J. Lehoczky, "Real-Time Synchronization Protocol for Multiprocessors," RTSS, 1988.

[20] J.A. Stankovic, T. He, T.F. Abdelzaher, M. Marley, G. Tao, S.H. Son, and C. Lu, "Feedback Control Real-Time Scheduling in Distributed Real-Time Systems," RTSS, 2001.

[21] D.C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole, "A Feedback-driven Proportion Allocator for Real-Rate Scheduling," OSDI, Feb 1999.

[22] J. Sun and J.W.S. Liu, "Synchronization Protocols in Distributed Real-Time Systems," ICDCS, 1996.

[23] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A New Resource ReSerVation Protocol," IEEE Network, September 1993.

APPENDIX A. DETAILED STABILITY ANALYSIS OF THE EXAMPLE IN SECTION VI.C

Example: We now apply the stability analysis approach to the example system described in the end of Section V. 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 assigned to all terms are 1. The cost function can be transformed to the following formula in scalar form:

$$V(k) = \sum_{j=1}^{2}\sum_{i=1}^{2}(u_j(k+i\,|\,k) - ref_j(k+i\,|\,k))^2 + \sum_{j=1}^{3}(\Delta r_j(k) - \Delta r_j(k-1))^2 \tag{A-1}$$

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

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

For simplicity, we use u_i and Δr_i to represent u_i(k) and $\Delta r_\mathrm{i}(k)$, respectively, in the rest of this section. Substitute (A-2) and the reference trajectory (8) in (A-1), the cost function becomes

$$\begin{aligned} V(k) = {} & [u_1 + c_{11}\Delta r_1 + c_{21}\Delta r_2 - (B_1 - \lambda(B_1 - u_1))]^2 \\ & + [u_2 + c_{22}\Delta r_2 + c_{31}\Delta r_3 - (B_2 - \lambda(B_2 - u_2))]^2 \\ & + [u_1 + 2c_{11}\Delta r_1 + 2c_{21}\Delta r_2 - (B_1 - \lambda^2(B_1 - u_1))]^2 \\ & + [u_2 + 2c_{22}\Delta r_2 + 2c_{31}\Delta r_3 - (B_2 - \lambda^2(B_2 - u_2))]^2 \\ & + [\Delta r_1 - \Delta r_1(k-1)]^2 + [\Delta r_2 - \Delta r_2(k-1)]^2 + [\Delta r_3 - \Delta r_3(k-1)]^2 \end{aligned} \tag{A-3}$$

where $\lambda = e^{-T_s/T_{ref}}$. We then perform partial differentiation on $V(k)$ with respect to Δr_1, Δr_2 and Δr_3, respectively. The derivatives are set to zero to compute the control input vector $\boldsymbol{\Delta r}(k)$ that minimize the cost function. This gives us the following equations:

$$\begin{cases} (10c_{11}{}^2 + 2)\Delta r_1 + 10c_{11}c_{21}\Delta r_2 + O = 0 \\ (10c_{31}{}^2 + 2)\Delta r_3 + 10c_{22}c_{31}\Delta r_2 + P = 0 \\ 10c_{11}c_{21}\Delta r_1 + (10c_{21}{}^2 + 10c_{22}{}^2 + 2)\Delta r_2 + 10c_{22}c_{31}\Delta r_3 + Q = 0 \end{cases} \tag{A-4}$$

where

$$O = 6c_{11}u_1 - 6c_{11}B_1 + (2c_{11}\lambda + 4c_{11}\lambda^2)(B_1 - u_1) - 2\Delta r_1(k-1)$$
$$P = 6c_{31}u_2 - 6c_{31}B_2 + (2c_{31}\lambda + 4c_{31}\lambda^2)(B_2 - u_2) - 2\Delta r_3(k-1)$$
$$Q = 6c_{21}u_1 + 6c_{22}u_2 - 6c_{21}B_1 - 6c_{22}B_2 + (2c_{21}\lambda + 4c_{21}\lambda^2)(B_1 - u_1) + (2c_{22}\lambda + 4c_{22}\lambda^2)(B_2 - u_2) - 2\Delta r_2(k-1)$$

We compute **Δ*r***(*k*) by solving (A-4), and then substitute it to the actual system model (5-6). The closed-loop model is a function of the system gains (g_1, g_2).

$$u_1(k+1) = [g_1(\frac{2\lambda + 4\lambda^2}{6} + \frac{2c_{21}}{3}\frac{(12c_{21} - 2c_{21}(2\lambda + 4\lambda^2))}{32c_{21}^2 + 32c_{22}^2} - 1) + 1]u_1(k)$$
$$+ g_1\frac{2c_{21}}{3}\frac{(12c_{22} - 2c_{22}(2\lambda + 4\lambda^2))}{32c_{21}^2 + 32c_{22}^2}u_2(k) + M$$

$$u_2(k+1) = g_2\frac{2c_{22}}{3}\frac{(12c_{21} - 2c_{21}(2\lambda + 4\lambda^2))}{32c_{21}^2 + 32c_{22}^2}u_1(k)$$
$$+ [g_2(\frac{2\lambda + 4\lambda^2}{6} + \frac{2c_{22}}{3}\frac{(12c_{22} - 2c_{22}(2\lambda + 4\lambda^2))}{32c_{21}^2 + 32c_{22}^2} - 1) + 1]u_2(k) + N$$

M, *N* are independent of $u_1(k)$ or $u_2(k)$. Hence the matrix ***A*** in (17) is

$$A = \begin{bmatrix} g_1(\frac{c}{6} + \frac{2c_{22}a}{3} - 1) + 1 & g_1\frac{2c_{21}b}{3} \\ g_2\frac{2c_{22}a}{3} & g_2(\frac{c}{6} + \frac{2c_{22}b}{3} - 1) + 1 \end{bmatrix} \tag{A-5}$$

where $a = \dfrac{(12c_{21} - 2c_{21}c)}{32c_{21}^2 + 32c_{22}^2}$, $b = \dfrac{(12c_{22} - 2c_{22}c)}{32c_{21}^2 + 32c_{22}^2}$, and $c = 2\lambda + 4\lambda^2$.

Since ***A*** is not a function of c_{11} or c_{31}, the stability of the closed-loop system only depends on the values of c_{21} and c_{22}. This is because both T_1 and T_3 are *local* tasks, i.e., each of them only has one subtask and hence only runs on a single processor. The controller can adjust the rates of T_1 and T_3 to control the utilization on a processor without affecting the other one. Therefore, only the parameters of the end-to-end task, T_2, affect system stability.

The closed-loop system is stable if the eigenvalues of ***A*** locate inside the unit circle in the complex space. The eigenvalues of ***A*** are

$$\rho = \frac{\frac{c}{6}(g_1 + g_2) + \frac{2}{3}(c_{21}ag_1 + c_{22}bg_2) - g_1 - g_2 + 2 \pm w^{1/2}}{2}$$

where

$$w = (\frac{c-6}{6})(g_1 - g_2)^2 + \frac{4}{3}(c_{21}ag_1 - c_{22}bg_2)(\frac{c-6}{6})(g_1 - g_2) + \frac{4}{9}(c_{21}ag_1 + c_{22}bg_2)^2$$

Following Step 3, we can establish the condition in terms of (g_1, g_2) that guarantees the stability of the closed-loop system. For example, in the special case when $g_1 = g_2 = g$,

$$\begin{cases} \rho_1 = \frac{2\lambda^2 + \lambda - 3}{3} g + 1 \\ \rho_2 = \frac{2\lambda^2 + \lambda - 3}{4} g + 1 \end{cases} \tag{A-6}$$

To guarantee stability, we need to guarantee $-1 < \rho_1, \rho_2 < 1$. Substituting the values of λ, T_s, and T_{ref}, the stability condition of the closed-loop system is $0 < g < 5.95$. Therefore, EUCON can maintain stability even if the execution time of every subtask becomes as high as 5.95 times its estimated one.

APPENDIX B. PARAMETERS OF THE MEDIUM TASK SET IN SECTION VII.A

The execution time of every subtask T_{ij} in MEDIUM follows a uniform distribution in a range [Min_{ij}, Max_{ij}]*etf, where *etf* is the current execution time factor used in the experiment. The second column (*Proc*) represents the processor where a subtask is located.

T_{ij}	***Proc***	Min_{ij}	Max_{ij}	$1/R_{max,i}$	$1/R_{min,i}$	$1/r_i(0)$	***phase***
T_{11}	P_1	25	35	55	3000	300	0
T_{12}	P_2	45	55				
T_{13}	P_3	45	55				
T_{14}	P_4	35	45				
T_{21}	P_4	45	55	55	5000	500	100
T_{22}	P_2	35	45				
T_{31}	P_1	55	65	65	4000	400	0
T_{32}	P_3	35	45				
T_{41}	P_1	25	35	45	6000	600	200
T_{42}	P_4	35	45				
T_{43}	P_2	15	25				
T_{51}	P_4	105	115	115	10000	1000	200
T_{52}	P_2	65	75				
T_{53}	P_3	55	65				
T_{61}	P_1	25	35	55	4000	400	0
T_{62}	P_2	45	55				
T_{63}	P_1	35	45				
T_{71}	P_4	55	65	105	6000	600	100
T_{72}	P_3	95	105				
T_{81}	P_2	65	75	75	5000	500	0
T_{82}	P_1	35	45				
T_{91}	P_1	35	45	45	5000	500	0
$T_{10,1}$	P_2	35	45	45	6000	600	0
$T_{11,1}$	P_3	35	45	45	4000	400	0
$T_{12,1}$	P_4	35	45	45	6500	650	0