DEUCON: Decentralized End-to-End Utilization Control for Distributed Real-Time Systems

Xiaorui Wang, *Member*, *IEEE*, Dong Jia, Chenyang Lu, *Member*, *IEEE*, and Xenofon Koutsoukos, *Member*, *IEEE*

Abstract—Many real-time systems must control their CPU utilizations in order to meet end-to-end deadlines and prevent overload. Utilization control is particularly challenging in distributed real-time systems with highly unpredictable workloads and a large number of end-to-end tasks and processors. This paper presents the Decentralized End-to-end Utilization CONtrol (DEUCON) algorithm, which can dynamically enforce the desired utilizations on multiple processors in such systems. In contrast to centralized control schemes adopted in earlier works, DEUCON features a novel decentralized control structure that requires only localized coordination among neighbor processors. DEUCON is systematically designed based on recent advances in distributed model predictive control theory. Both control-theoretic analysis and simulations show that DEUCON can provide robust utilization guarantees and maintain global system stability despite severe variations in task execution times. Furthermore, DEUCON can effectively distribute the computation and communication cost to different processors and tolerate considerable communication delay between local controllers. Our results indicate that DEUCON can provide a scalable and robust utilization control for large-scale distributed real-time systems executing in unpredictable environments.

Index Terms—Real-time and embedded systems, feedback control real-time scheduling, distributed systems, end-to-end task, decentralized model predictive control.

1 Introduction

Recent years have seen the rapid growth of Distributed Real-time Embedded (DRE) applications executing in *unpredictable* environments in which workloads are unknown and vary significantly at runtime. Such systems include data-driven and open systems whose execution is heavily influenced by volatile environments. For example, task execution times in vision-based feedback control systems depend on the content of live camera images of changing environments [13]. Likewise, supervisory control and data acquisition (SCADA) systems for power grid control may experience a dramatic load increase during a cascade power failure [9]. Furthermore, as DRE systems become connected to the Internet, they are exposed to load disturbances due to variable user requests and even cyber attacks [9]. As such systems become increasingly important to our society, a new paradigm of real-time computing based on *Adaptive Quality-of-service (QoS) Control (AQC)* has received significant attention. In contrast to traditional approaches to real-time systems that rely on accurate knowledge about the system workload, AQC can provide robust QoS guarantees in unpredictable environments by adapting to workload variations based on dynamic feedback. A key advantage of AQC is that it adopts a control-theoretic framework for systematically developing adaptation strategies. This rigorous design methodology is in sharp contrast to heuristic-based adaptive solutions that rely on extensive empirical evaluation and manual tuning.

In this paper, we focus on *utilization control*, which is an important instance of AQC for distributed soft real-time systems. The goal of utilization control is to enforce the desired CPU utilizations on all the processors in a distributed system despite significant uncertainties in system workloads. Utilization control can be used to enforce appropriate schedulable utilization bounds on all processors to guarantee end-to-end task deadlines. It can also enhance system survivability by providing overload protection against workload fluctuation.

Our proposed framework can be beneficial to DRE applications that are amenable to rate adaptation such as digital feedback control systems [25], [28], monitoring systems [36], and multimedia [4]. Utilization control in these systems can be performed by adjusting the task rates. Tasks running at higher rates contribute higher values to the application (for example, increasing the sampling rate of a digital controller improves the control performance). Furthermore, task rates can be adjusted dynamically without causing system failure. Decentralized AQC approaches, in particular, are valuable for SCADA systems that include multiple geographically distributed monitoring and control subsystems [9], [11].

- *X. Wang is with the Department of Electrical and Computer Engineering, University of Tennessee, Knoxville, 421 Ferris Hall, 1508 Middle Drive, Knoxville, TN 37996-2100. E-mail: xwang@ece.utk.edu.*
- *D. Jia is with The Mathworks, 3 Apple Hill Drive, Natick, MA 01760. E-mail: djia@mathworks.com.*
- *C. Lu is with the Department of Computer Science and Engineering, Washington University in St. Louis, 1 Brookings Dr., Box 1045, St. Louis, MO 63130-4899. E-mail: lu@cse.wustl.edu.*
- *X. Koutsoukos is with the Department of Electrical Engineering and Computer Science, Institute for Software Integrated Systems (ISIS), Vanderbilt University, Box 1679, Station B, Nashville, TN 37235. E-mail: Xenofon.Koutsoukos@vanderbilt.edu.*

Manuscript received 30 Jan. 2006; revised 31 Aug. 2006; accepted 26 Oct. 2006; published online 9 Jan. 2007.
Recommended for acceptance by J. Hou.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0022-0106.
Digital Object Identifier no. 10.1109/TPDS.2007.1051.

DRE systems introduce many new research challenges that have not been addressed in earlier work on single-processor systems. First, they require *multiple-input-multiple-output (MIMO)* control solutions to manage the system QoS on multiple processors. Second, the QoS of different processors are often *coupled* with each other due to complex interactions among distributed application components. In particular, many DRE systems employ the common *end-to-end task model* [20], where a task may comprise a chain of subtasks on different processors. In such systems, the CPU utilizations of different processors cannot be controlled independently. For example, changing the rate of a task will affect the CPU utilizations of all the processors where its subtasks are located. Therefore, the coupling among processors must be modeled and addressed in the design of QoS control algorithms. Finally, a utilization control algorithm must be highly scalable in order to handle large DRE systems (for example, power grid management and smart spaces). A centralized control algorithm is often inadequate for such systems since its communication and computation overhead usually depends on the size of the *entire* DRE system.

In this paper, we present the *Decentralized* End-to-end Utilization CONtrol (*DEUCON*) algorithm for large DRE systems with end-to-end tasks. In sharp contrast to earlier solutions based on centralized control schemes [23], DEUCON employs a completely *decentralized* control approach that can scale well in large distributed systems and tolerate individual processor failures. Specifically, the contributions of this paper are fourfold:

- We propose a new approach for decomposing the global multiprocessor utilization control problem into local subproblems to facilitate the design of decentralized control solutions.
- We describe the DEUCON algorithm featuring a novel peer-to-peer control structure that enforces the desired utilizations of multiple processors through localized coordination among controllers.
- We give a control analysis based on the *distributed model predictive control* (DMPC) theory [8], which establishes the stability properties of the DEUCON algorithm in the face of uncertain task execution times.
- We present simulation results showing that DEUCON can provide robust utilization guarantees to multiple processors through task rate adaptation[1] while achieving scalability by effectively distributing the computation and communication overhead to local controllers.

The rest of this paper is organized as follows: Section 2 reviews related work. Section 3 formulates the end-to-end utilization control problem. Section 4 describes an existing centralized utilization control algorithm as a starting point for this work. Section 5 presents the design and analysis of DEUCON. Section 6 evaluates DEUCON with simulations. The paper concludes with Section 7.

1. Our approach may be extended to exploit other control mechanisms such as quality level adaptation and dynamic voltage scaling that can be used to change task execution times online.

2 Related Work

Traditional approaches for handling end-to-end tasks such as end-to-end scheduling [33] and distributed priority ceiling [27] rely on schedulability analysis, which requires a priori knowledge of 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 execution times is resource reclaiming [6], [30]. A drawback of existing resource-reclaiming techniques is that they often require modifications to low-level scheduling mechanisms in operating systems. In contrast, the feedback control approach and rate adaptation techniques adopted in this paper can be easily implemented at the application or middleware layer on top of standard platforms [22].

Control-theoretic approaches have been applied to a number of computing systems. A survey of feedback performance control in computing systems is presented in [1]. Projects that applied control theory to real-time scheduling and utilization control are directly related to this paper. Steere et al. and Goel et al. developed feedback-based schedulers [12], [32] that guarantee the desired progress for real-time applications. Abeni et al. presented a control analysis of a reservation-based feedback scheduler [2]. Zhu and Mueller applied feedback control scheduling to processor power control [37]. Cervin et al. proposed a feedback-feedforward scheduler specialized for digital control applications [10], [29]. All the aforementioned projects focused on controlling the performance of *single*-processor systems. Their algorithms are based on single-input-single-output linear control techniques that are not applicable to DRE systems with multiple processors.

Stankovic et al. and Lin and Manimaran proposed feedback control scheduling algorithms for DRE systems with *independent* tasks [31], [18]. Both papers presented hierarchical control structures that combine decentralized control with local control algorithms. However, neither handled the dependencies among processors caused by end-to-end tasks commonly available in DRE systems. Handling the dependencies with decentralized control represents a major research challenge, which is addressed by the design of DEUCON.

DEUCON builds on our earlier work on feedback control real-time scheduling. Lu et al. [21] proposed feedback control scheduling algorithms for controlling the CPU utilization and deadline miss ratio on a single processor. These algorithms have been implemented as a middleware service and integrated with a real-time Object Request Broker middleware called FCS/nORB [22]. Koutsoukos et al. proposed another single-processor control algorithm that can handle discrete control inputs such as a finite set of task rates [15]. Our more recent work extended feedback control real-time scheduling to DRE systems. Lu et al. developed the End-to-end Utilization CONtrol (EUCON) [23], the first utilization control algorithm designed for DRE systems with end-to-end tasks. EUCON manages and coordinates the adaptation of multiple processors with a *centralized* controller that cannot scale effectively in large-scale DRE systems. We discuss EUCON in more detail in Section 5. Wang et al. implemented an extended version of EUCON in a DRE middleware called

FC-ORB [34]. FC-ORB features a unified architecture that integrates EUCON, end-to-end real-time scheduling, and fault tolerance mechanisms to enhance the robustness of DRE applications. FC-ORB also employs a centralized control architecture with a single controller. To our knowledge, DEUCON is the first *decentralized* control algorithm designed for DRE systems with end-to-end tasks.

3 End-to-End Utilization Control

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

3.1 Task Model

We adopt an end-to-end task model [20] implemented by many DRE applications. The system is comprised of m periodic tasks $\{T_i|1 \leq i \leq m\}$ executing on n processors $\{P_i|1 \leq i \leq n\}$. Task T_i is composed of a set of subtasks $\{T_{ij}|1 \leq j \leq n_i\}$ that may be located on different processors. A processor may host one or more subtasks of a task. The release of subtasks is subject to precedence constraints, that is, subtask $T_{ij}(1 < j \leq n_i)$ cannot be released for execution until its predecessor subtask T_{ij-1} is completed. All the subtasks of a task share the same rate. The rate of a task (and all its subtasks) can be adjusted by changing the rate of its first subtask. If a nongreedy synchronization protocol is used (for example, release guard [33]), every subtask is released periodically without jitter. The processor P_j hosting the first subtask of a task T_i is called T_i's *master processor* and we say that P_j *masters* T_i. Only a task's master processor can change its rate.

Our task model has three important properties. First, although each subtask T_{ij} has an *estimated* execution time c_{ij} available at design time, its *actual* execution time may be different from its estimation and vary at runtime. Modeling such uncertainty is important to DRE systems operating in unpredictable environments. Second, the rate of a task T_i may be dynamically adjusted within a range $[R_{min,i}, R_{max,i}]$. This assumption is based on the fact that the task rates in many applications (for example, digital control [25], [28], sensor update, and multimedia [4], [5]) can be dynamically adjusted without causing system failure. A task running at a higher rate contributes a higher value to the application at the cost of higher utilizations. Third, each task T_i has a *soft* end-to-end deadline related to its period. In an end-to-end scheduling approach [33], the deadline of an end-to-end task is divided into subdeadlines of its subtasks [14], [26]. Hence, the problem of meeting the deadline can be transformed to the problem of meeting the subdeadline of each subtask. A well-known approach for meeting the subdeadlines on a processor is to ensure that its utilization remains below its schedulable utilization bound [16], [19].

3.2 Problem Formulation

Utilization control can be formulated as a dynamic constrained optimization problem. We first introduce several notations. T_s, the sampling period, is selected so that multiple instances of each task may be released during a sampling period. $u_i(k)$ is the CPU utilization of processor P_i in the kth sampling period, that is, the fraction of time that P_i is not idle during time interval $[(k-1)T_s, kT_s)$. B_i is the desired utilization setpoint on P_i. $r_j(k)$ is the invocation rate of task T_j in the $(k+1)$th sampling period.

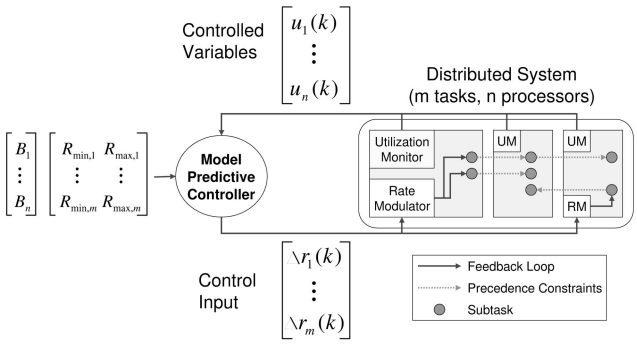

Fig. 1. EUCON's feedback control loop with a centralized controller.

Given a utilization setpoint vector $\mathbf{B} = [B_1 \dots B_n]^T$ and rate constraints $[R_{min,j}, R_{max,j}]$ for each task T_j, the control goal at the kth sampling point (time kT_s) is to dynamically choose task rates $\{r_j(k)|1 \leq j \leq m\}$ to minimize the difference between B_i and $u_i(k)$ for all processors:

$$\min_{\{r_j(k)|1 \leq j \leq m\}} \sum_{i=1}^{n} (B_i - u_i(k+1))^2 \quad (1)$$

subject to constraints

$$R_{min,j} \leq r_j(k) \leq R_{max,j} \quad (1 \leq j \leq m).$$

The rate constraints ensure that all tasks remain within their acceptable rate ranges. The optimization formulation maximizes task rates by making the utilization of each processor as close to its setpoint as allowed by the constraints. The design goal is to ensure that all processors quickly converge to their utilization setpoints after a workload variation whenever it is feasible under the rate constraints. Therefore, to guarantee end-to-end deadlines, a user only needs to specify the setpoint of each processor to be a value below its schedulable utilization bound. Utilization control algorithms can be used to meet all the end-to-end deadlines by enforcing the setpoints of all the processors in a DRE system when feasible under the rate constraints.[2]

4 EUCON: A Centralized Algorithm

In this section, we briefly describe the EUCON algorithm [23], which provides a starting point and baseline for our work.

As shown in Fig. 1, EUCON features a feedback control loop composed of a centralized model predictive controller (MPC) and a utilization monitor and rate modulator on each processor. EUCON is invoked periodically at each sampling point k. The controlled variables are the utilizations of all processors, $\mathbf{u(k)} = [u_1(k) \dots u_n(k)]^T$. The control inputs from the controller are the change in task rates $\mathbf{\Delta r(k)} = [\Delta r_1(k) \dots \Delta r_m(k)]^T$, where $\Delta r_i(k) = r_i(k) - r_i(k-1)$ $(1 \leq i \leq m)$.

The feedback control loop works as follows: 1) the utilization monitor on each processor P_i sends its utilization

2. A system must apply admission control when its load exceeds the limit that can be handled within the rate constraints.

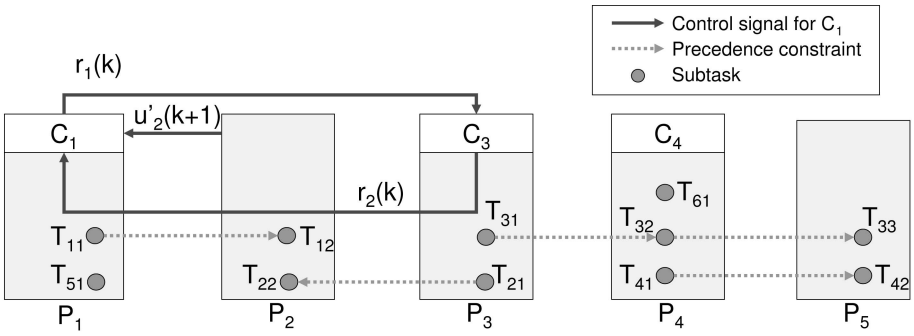

Fig. 2. Data exchange between C_1 and its neighbors (other data exchanges are not shown).

$u_i(k)$ in the last sampling period $[(k-1)T_s, kT_s)$ to the centralized controller, 2) the controller collects the utilization vector $\mathbf{u(k)} = [u_1(k) \dots u_n(k)]^T$, including the utilizations of all processors, computes a new rate change vector $\mathbf{\Delta r(k)} = [\Delta r_1(k) \dots \Delta r_m(k)]^T$, and sends the new task rates $\mathbf{r(k)} = \mathbf{r(k-1)} + \mathbf{\Delta r(k)}$ to the rate modulators on master processors (that is, processors that master at least one task), and 3) the rate modulators on the master processors change the rates of tasks according to $\mathbf{r(k)}$. The details of the controller design in EUCON are described in [23].

EUCON relies on a centralized controller to manage the adaptation of multiple processors in a DRE system. A centralized control scheme has several disadvantages. First, the runtime overhead depends on the size of an entire DRE system. Specifically, the worst-case computational complexity of an MPC is polynomial in the total number of tasks and the total number of processors in the system. Furthermore, since every processor in the system needs to communicate with the controller in every sampling period, the processor executing the controller can become a communication bottleneck. Therefore, a centralized control scheme cannot scale effectively in large DRE systems. Second, the control design of EUCON assumes that communication delays between the control processor and other processors are negligible compared to the sampling period of the controller. This assumption may not hold in networks with significant delays such as the Internet and wireless sensor networks. In addition, the processor executing the controller is a single point of failure. The entire system will lose the capability to adapt to the environment if it fails.

Centralized solutions are therefore not suitable for large-scale DRE systems (for example, wide-area power grid management). In this paper, we focus on developing *decentralized* control algorithms to improve the scalability and reliability of adaptive utilization control in DRE systems.

5 Design of DEUCON

In contrast to the centralized control scheme adopted by EUCON, DEUCON employs a peer-to-peer control structure with a separate local controller C_i on each master processor P_i. Each controller only coordinates with a small number of processors called its (logical) *neighbors*. A fundamental design challenge is to achieve system stability and the desired utilizations without global information. In this section, we present the design of DEUCON based on a DMPC framework. As a foundation of our control design, we first present a dynamic model of the entire system and an approach for decomposing the global system model into localized control subproblems. We then describe the design and control analysis of the DEUCON algorithm based on the dynamic models.

5.1 Global System Model

In a control-theoretic methodology, a control algorithm should be designed based on a model of the system. As described in [23], a DRE system can be approximated by the following *global* system model:

$$\mathbf{u(k+1)} = \mathbf{u(k)} + \mathbf{G(k)F\Delta r(k)}. \quad (2)$$

The vector $\mathbf{\Delta r(k)}$ represents the changes in task rates. The *subtask allocation matrix* $\mathbf{F}$ is an $n \times m$ matrix, where $f_{ij} = \sum_{c_{jl} \in S_{ij}} c_{jl}$ if one or more subtasks of task T_j are allocated to processor P_i and $f_{ij} = 0$ if no subtask of task T_j is allocated to processor P_i. S_{ij} is the set of subtasks of T_j located on processor P_i. $\mathbf{F}$ captures the *coupling* among processors due to end-to-end tasks. $\mathbf{G(k)} = diag[g_1(k) \dots g_n(k)]$, where $g_i(k)$ represents the ratio between the change in the actual utilization and its estimation. The exact value of $g_i(k)$ is *unknown* due to the unpredictability in execution times. Note that $\mathbf{G(k)}$ describes the effect of uncertainty in workload on the utilization of a DRE system. As an example, Fig. 2 shows a DRE system with five processors and six tasks. It is modeled by (2) with the following parameters:

$$\mathbf{u(k)} = \begin{bmatrix} u_1(k) \\ u_2(k) \\ u_3(k) \\ u_4(k) \\ u_5(k) \end{bmatrix}, \ \mathbf{F} = \begin{bmatrix} c_{11} & 0 & 0 & 0 & c_{51} & 0 \\ c_{12} & c_{22} & 0 & 0 & 0 & 0 \\ 0 & c_{21} & c_{31} & 0 & 0 & 0 \\ 0 & 0 & c_{32} & c_{41} & 0 & c_{61} \\ 0 & 0 & c_{33} & c_{42} & 0 & 0 \end{bmatrix},$$

$$\mathbf{\Delta r(k)} = \begin{bmatrix} \Delta r_1(k) \\ \Delta r_2(k) \\ \Delta r_3(k) \\ \Delta r_4(k) \\ \Delta r_5(k) \\ \Delta r_6(k) \end{bmatrix}, \ \mathbf{G(k)} = \begin{bmatrix} g_1(k) & 0 & 0 & 0 & 0 \\ 0 & g_2(k) & 0 & 0 & 0 \\ 0 & 0 & g_3(k) & 0 & 0 \\ 0 & 0 & 0 & g_4(k) & 0 \\ 0 & 0 & 0 & 0 & g_5(k) \end{bmatrix}.$$

5.2 Problem Decomposition

Although our previous work showed that the above global system model is sufficient for designing a centralized

controller for EUCON [23], it cannot be used for designing decentralized control algorithms because it includes information about the entire system. To address this problem, we propose a new approach to decompose the global utilization control problem into a set of localized subproblems.

From a local controller C_i's perspective, the goal of decomposition is to partition the set of system variables into three subsets, including *local variables* on host processor P_i, *neighbor variables* on P_i's neighbors, and all other variables in the system. C_i's subproblem only includes its local and neighbor variables. A key feature of our decomposition scheme is that it balances two conflicting goals. On one hand, the number of neighbor variables should be minimized to improve system scalability. On the other hand, the neighbor variables must capture the coupling among processors so that local controllers can achieve global system stability through coordination in their neighborhoods.

We give several definitions before presenting our decomposition scheme.

Definition 1. *Processor P_j is P_i's* direct neighbor *if 1) P_j has a subtask belonging to an end-to-end task mastered by P_i and 2) P_j is not P_i itself.*

Definition 2. *The* concerned tasks *of P_i are the tasks that have subtasks located on P_i or P_i's direct neighbors.*

Definition 3. *Processor P_j is P_i's* indirect neighbor *if 1) P_j is the master processor of any of P_i's concerned tasks and 2) P_j is not P_i's direct neighbor or P_i itself.*

For example, we consider controller C_1 in the system shown in Fig. 2. P_1 has one direct neighbor (P_2) due to task T_1 mastered by P_1. Its concerned tasks include T_1, T_5, and T_2 (which has a subtask on direct neighbor P_2). Hence, P_3, the master processor of T_2, is P_1's indirect neighbor.

The subproblem of a controller includes a set of utilizations as *controlled variables* and a set of task rates as *manipulated variables*. In our decomposition scheme, the controlled variables of controller C_i include $u_i(k)$, the host processor P_i's utilization, and $UD_i(k)$, the set of utilizations of P_i's direct neighbors. $UD_i(k)$ are considered C_i's neighbor variables because they are affected by the rates of tasks mastered by P_i. Since each concerned task contributes to the utilizations of P_i and/or its direct neighbors, C_i's manipulated variables include the rates of all of P_i's concerned tasks. Note that a concerned task may be mastered by P_i itself, its direct neighbor, or its indirect neighbor. For example, C_1 has two controlled variables, $u_1(k)$ and $u_2(k)$, and three manipulated variables, $r_1(k)$, $r_2(k)$, and $r_5(k)$.

Let set $NR_i(k)$ denote the rates of all of P_i's concerned tasks and set $NU_i(k) = UD_i(k) \cup \{u_i(k)\}$. The subproblem of C_i then becomes the following localized constrained optimization problem within its neighborhood:

$$\min_{NR_i(k)} \sum_{u_l(k) \in NU_i(k)} (B_l - u_l(k+1))^2 \tag{3}$$

subject to

$$R_{min,j} \le r_j(k) \le R_{max,j} \quad (r_j(k) \in NR_i(k)).$$

In contrast to the global model (2) used in EUCON, each controller in DEUCON has a localized model that only includes its local and neighbor variables. This local model of C_i is described as

$$\mathbf{nu_i(k+1)} = \mathbf{nu_i(k)} + \mathbf{G_i(k)F_i\Delta nr_i(k)}, \tag{4}$$

where $\mathbf{nu_i(k)}$ and $\mathbf{nr_i(k)}$ are vectors comprised of all elements in $NU_i(k)$ and $NR_i(k)$, respectively. $\mathbf{G_i(k)}$ and $\mathbf{F_i}$ are defined in the same way as $\mathbf{G(k)}$ and $\mathbf{F}$ in (2) but include only the processors in $NU_i(k)$ and the task rates in $NR_i(k)$.

For example, the controller C_1 shown in Fig. 2 is modeled with the following parameters:

$$\mathbf{nu_1(k)} = \begin{bmatrix} u_1(k) \\ u_2(k) \end{bmatrix}, \mathbf{F_1} = \begin{bmatrix} c_{11} & 0 & c_{51} \\ c_{12} & c_{22} & 0 \end{bmatrix},$$

$$\mathbf{G_1(k)} = \begin{bmatrix} g_1(k) & 0 \\ 0 & g_2(k) \end{bmatrix}, \mathbf{\Delta nr_1(k)} = \begin{bmatrix} \Delta r_1(k) \\ \Delta r_2(k) \\ \Delta r_5(k) \end{bmatrix}.$$

From (4), C_1's local model is

$$\begin{aligned} u_1(k+1) &= u_1(k) + g_1(k)(c_{11}\Delta r_1(k) + c_{51}\Delta r_5(k)), \\ u_2(k+1) &= u_2(k) + g_2(k)(c_{12}\Delta r_1(k) + c_{22}\Delta r_2(k)). \end{aligned}$$

5.3 Localized Feedback Control Loop

We now present DEUCON's localized feedback control loop based on our decomposition scheme. The execution of a controller C_i at each sampling point k includes three steps:

1. *Local control computation*. C_i executes an MPC algorithm to solve its local subproblem. The feedback input to the control algorithm includes 1) $u_i(k)$ from the local utilization monitor, 2) a set of *predicted utilizations* $UD_i'(k)$ of its direct neighbors, and 3) the rates of concerned tasks $NR_i(k-1)$ in the last sampling period. The output from the controller C_i includes the new rates for concerned tasks $NR_i(k)$. The details of the control algorithm are presented in Section 5.4.
2. *Local actuation*. The rate modulator on P_i changes the rates of the set of tasks mastered by P_i according to the control input from C_i. The other task rates in the control input will be ignored because they are not mastered by P_i.
3. *Data exchange among neighbors*. C_i sends its *predicted utilization* at the next sampling point $u_i'(k+1)$ to other controllers of which it serves as a direct neighbor. C_i also sends the rates of tasks mastered by P_i to those controllers that have these tasks as their concerned tasks. In addition, C_i receives new predicted utilizations from its direct neighbors, and the actual rates of the concerned tasks that are not mastered by itself, from its direct and indirect neighbors. They will be used for the local control computation at the next sampling point $(k+1)$.

Compared to centralized control schemes, a fundamental advantage of DEUCON is that both the computation and communication overhead of a controller depend on the size of its neighborhood instead of the entire system. This

feature allows DEUCON to scale effectively in many large DRE systems.

Another important advantage of DEUCON is that it can tolerate considerable network delays. Note that, in Step 1, the *predicted* utilizations $UD'_i(k)$ (instead of $UD_i(k)$) are provided by C_i's direct neighbors in the previous sampling period. This is because $UD_i(k)$ is not instantaneously available to C_i at time kT_s due to network delays. $UD'_i(k)$ is predicted based on $UD_i(k-1)$ at time $(k-1)T_s$ as a substitute for $UD_i(k)$ to be transmitted over the network during interval $[(k-1)T_s, kT_s)$. Each element $u'_j(k) \in UD'_i(k)$ is calculated using the following reference trajectory from measured utilization $u_j(k-1)$ to its setpoint B_j over the following P sampling periods:

$$ref_j((k-1)+l|k-1) = B_j - e^{-\frac{T_s}{T_{ref}}l}(B_j - u_j(k-1)) \quad (1 \le l \le P), \tag{5}$$

where T_{ref} is the time constant that specifies the speed of system response. P is called the *prediction horizon*. The notation $x((k-1)+l|k-1)$ means that the value of variable x at time $((k-1)+l)T_s$ depends on the conditions at time $(k-1)T_s$. The value of $ref_j(k|k-1)$ is assigned to $u'_j(k)$. Since $UD'_i(k)$ can take the whole last sampling period to transmit, DEUCON can tolerate much longer communication delays than EUCON, which assumes the delays to be negligible. This approach can be easily extended to handle larger communication delays. For example, if the transmission needs to take two sampling periods, $ref_j(k|k-2)$ can be assigned to $u'_j(k)$. It is not necessary for all processors to predict the same steps into the future. Similarly, each processor does not need to send the same prediction to different neighbors. For instance, they could send a one-step prediction $ref_j(k|k-1)$ to one-hop neighbors, a two-step prediction $ref_j(k|k-2)$ to two-hop neighbors, and so on if the communication delay for one-hop is one sampling period.

DEUCON can also improve system fault-tolerance by avoiding a centralized controller, which is a single point of failure in the whole system. In DEUCON, even if the system failure of a processor may disable a local controller, the subtasks on the failed processor can be immediately migrated to their backup processors and then be effectively controlled by other local controllers there. As a result, single processor failures will not cause the system to lose control in DEUCON.

5.4 Controller Design

DEUCON employs a local controller on each *master* processor. Nonmaster processors do not need controllers because they cannot change the rate of any task. For the example shown in Fig. 2, processors P_1, P_3, and P_4 each have a controller, whereas P_2 and P_5 do not have controllers because they are not master processors for any tasks. This feature reduces the overhead of DEUCON.

We design a model predictive control algorithm [7] for controller C_i. We choose model predictive control because it can deal with coupled MIMO control problems with constraints on the actuators. At every sampling point, the controller computes an input trajectory in the following M sampling periods, for example,

$$\mathbf{\Delta nr_i(k)}, \mathbf{\Delta nr_i(k+1|k)}, \ldots \mathbf{\Delta nr_i(k+M-1|k)},$$

which minimizes the following cost function under the rate constraints:

$$V_i(k) = \sum_{l=1}^{P} \|\mathbf{nu_i(k+l|k)} - \mathbf{ref_i(k+l|k)}\|^2 + \sum_{l=0}^{M-1} \|\mathbf{\Delta nr_i(k+l|k)} - \mathbf{\Delta nr_i(k+l-1|k)}\|^2, \tag{6}$$

where P is the *prediction horizon* and M is the *control horizon*. The first term in the cost function represents the *tracking error*, that is, the difference between the utilization vector $\mathbf{nu_i(k+l|k)}$, which is predicted based on (7) and the reference trajectory $\mathbf{ref_i(k+l|k)}$ defined in (5). The controller is designed to track the exponential reference trajectory that converges to the setpoints so that the closed-loop system behaves like a desired linear system. By minimizing the tracking error, the closed-loop system will also converge to the utilization setpoints. The second term in the cost function represents the *control penalty*. The control penalty term causes the controller to minimize the changes in the control input.

The controller predicts the cost based on the following *approximate* model:

$$\mathbf{nu_i(k+1)} = \mathbf{nu'_i(k)} + \mathbf{F_i \Delta nr_i(k)}. \tag{7}$$

The above model has two differences from the *actual* system model (4). First, the utilizations of direct neighbors are approximated by their predicted utilizations $\mathbf{nu'_i(k)}$, where $\mathbf{nu'_i(k)}$ is a vector comprised of all elements in $NU'_i(k)$. As discussed in Section 5.3, this approximation allows DEUCON to tolerate network delays. Second, because the real system gains $\mathbf{G_i(k)}$ in system model (4) are unknown in unpredicted environments, our controller assumes that $\mathbf{G_i(k)} = diag[1 \ldots 1]$, that is, the controller assumes that the estimated execution times are accurate. Although this approximate model is not an exact characterization of the real system, the closed-loop system under our controller can still maintain stability and guarantee the desired utilization setpoints as long as $\mathbf{G_i(k)}$ are within a certain range (see analysis and simulation results in Sections 5.5 and 6.2). This is due to the coordination scheme and the online feedback controls used in our DMPC algorithm.

The controller computes the input trajectory

$$\mathbf{\Delta nr_i(k)}, \mathbf{\Delta nr_i(k+1|k)}, \ldots \mathbf{\Delta nr_i(k+M-1|k)},$$

which minimizes the cost function subject to the rate constraints. This constrained optimization problem can be transformed to a standard constrained least squares problem [24], [23]. Controller C_i can then use a standard least squares solver to solve this problem online. The detailed transformation is not shown due to space limitations. The worst-case computation complexity of the solver is polynomial in the numbers of tasks and processors in the localized model (7). More specifically, our constrained least squares optimization is a convex nonlinear optimization for which interior point methods require $O(n)$ Newton iterations [35], where n is the number of optimization variables. Since each Newton iteration requires $O(n^3)$ algebraic operations, the worst-case computation complexity of the

solver is cubic in the number of tasks and processors in the localized model.

Once the input trajectory is computed, only the first element $\Delta\mathbf{nr_i}(\mathbf{k})$ is applied as the control input and sent to the rate modulators. At the next sampling point, the prediction horizon slides one sampling period and the control input is computed again as a solution to the constrained optimization problem based on the utilization feedbacks from its direct neighbors and itself.

DEUCON has three tunable parameters: T_{ref}, P, and M. The time constant T_{ref} reflects how fast we expect the system to converge to the setpoint. The smaller the value T_{ref} takes, the faster the system is expected to converge to the setpoint value. However, a too small value may result in a large overshoot. Thus, T_{ref} should be chosen as a trade-off between speed and overshoot. The prediction horizon P, in general, should be large enough such that the reference trajectory converges to the setpoint value and the system converges to the reference trajectory. However, the larger P is, the more computation load there is. The control horizon M affects the freedom for the least squares solver to change Δnr_i. The more freedom the solver has, the better solution it can generate. As the value of M increases, the dimension of the solution space increases and, therefore, searching the optimal solution takes more time. In addition, due to the decentralized nature of DEUCON, P should not take too large of a value since the control decisions are based on the prediction of the future states that will be affected by the prediction of the neighbors. The deviation of the predicted utilization trajectory from the real trajectory increases along the prediction horizon. For large P, the prediction error of the utilization in the end of the prediction horizon may be high, resulting in poor control decisions. Therefore, the choice of the parameters must balance the prediction error, quality of the solution, and computation load.

5.5 Stability Analysis

A fundamental benefit of the control-theoretic approach is that it enables us to prove the utilization guarantees provided by DEUCON despite uncertainties in task execution times. We say that a DRE system is *stable* if the utilizations $\mathbf{u}$ converge to the desired setpoints $\mathbf{B}$, that is, $\lim_{\mathbf{k}\to\infty}\mathbf{u}(\mathbf{k}) = \mathbf{B}$. In this section, we present a method that, given a system and a range of variations in task execution times, allows to analytically assess the stability and robustness of DEUCON. To ensure that the system can be stabilized, the constrained optimization problem must be feasible, that is, there exists a set of task rates within their acceptable ranges that can make the utilization on every processor equal to its setpoint. If the problem is infeasible, no controller can guarantee the setpoint through rate adaptation. In this case, the system may switch to a different control adaptation mechanism (for example, admission control or task reallocation). Henceforth, our stability analysis assumes that the rate constraints are not activated.

In DEUCON, each controller solves a finite-horizon optimal tracking problem. Based on optimal control theory [17], the local control decision is a linear function of the current utilization and the setpoint of the local CPU, the utilizations of its direct neighbors, and the previous decisions for its manipulated tasks and concerned tasks. We now outline the process for analyzing the stability of the system controlled by DEUCON:

1. Compute the feedback and feedforward matrices for each local controller i by solving its local control input $\Delta\mathbf{nr_i}$ based on the local approximate system model (7) and reference trajectory (5). The solution is in the following form:
$$\Delta\mathbf{nr_i}(\mathbf{k}) = \mathbf{K_i}\mathbf{nu_i'}(\mathbf{k}) + \mathbf{H_i}\Delta\mathbf{nr_i}(\mathbf{k-1}) + \mathbf{E_i}\mathbf{B_i}. \quad (8)$$
2. Construct the feedback and feedforward matrices for the whole system (2) based on those for local system models derived in Step 1:
$$\Delta\mathbf{r}(\mathbf{k}) = \mathbf{K}\mathbf{u}(\mathbf{k}) + \mathbf{L}\mathbf{u}(\mathbf{k-1}) + \mathbf{H}\Delta\mathbf{r}(\mathbf{k-1}) + \mathbf{E}\mathbf{B}. \quad (9)$$
This is a dynamic controller. The stability analysis needs to consider the composite system consisting of the dynamics of the original system and the controller.
3. Derive the closed-loop model of the composite system by substituting the control inputs derived in Step 2 into the *actual* system model described by (2). The closed-loop composite system is in the following form:
$$\begin{bmatrix}\mathbf{u}(\mathbf{k+1})\\ \mathbf{u}(\mathbf{k})\\ \Delta\mathbf{r}(\mathbf{k})\end{bmatrix} = \begin{bmatrix}\mathbf{I}+\mathbf{G}(\mathbf{k})\mathbf{F}\mathbf{K} & \mathbf{G}(\mathbf{k})\mathbf{F}\mathbf{L} & \mathbf{G}(\mathbf{k})\mathbf{F}\mathbf{H}\\ \mathbf{I} & \mathbf{0} & \mathbf{0}\\ \mathbf{K} & \mathbf{L} & \mathbf{H}\end{bmatrix} \begin{bmatrix}\mathbf{u}(\mathbf{k})\\ \mathbf{u}(\mathbf{k-1})\\ \Delta\mathbf{r}(\mathbf{k-1})\end{bmatrix} + \begin{bmatrix}\mathbf{G}(\mathbf{k})\mathbf{F}\mathbf{E}\\ \mathbf{0}\\ \mathbf{E}\end{bmatrix}\mathbf{B}, \quad (10)$$
where $\mathbf{I}$ is the identity matrix. Note that the closed-loop system model is a function of $\mathbf{G}$.
4. Derive the stability condition of the closed-loop system (10) given a range of $\mathbf{G}$ values. According to control theory, for a time-invariant $\mathbf{G}$, if all poles are located inside the unit circle in the complex space and the DC-gain matrix from the control input $\Delta\mathbf{r}(\mathbf{k})$ to the system state $\mathbf{u}(\mathbf{k})$ is the identity matrix, the utilizations $\mathbf{u}(\mathbf{k})$ will converge to the setpoints. For a time-varying $\mathbf{G}$, if there exists a Lyapunov function for the closed-loop system (10) for all $\mathbf{G}$s in a range, then the system is stable if $\mathbf{G}$ varies within that range.

The details of the above steps follow the method given in [8] and are not shown due to space limitations. We have developed a Matlab program to perform the above stability analysis procedure automatically.

Example. To illustrate our method for stability analysis, we now apply the stability analysis approach to the example system described in Fig. 3. The system has 21 tasks and 10 processors. We set the prediction horizon $P = 2$ and the control horizon $M = 1$. The time constant of the

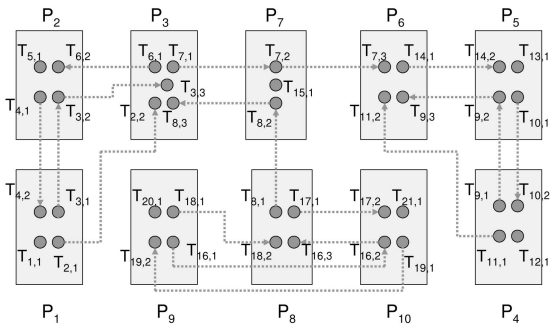

Fig. 3. A medium-sized workload.

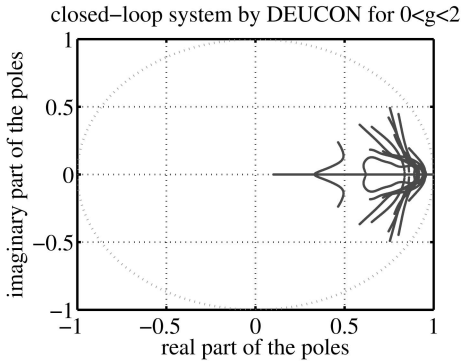

Fig. 4. The root locus of the closed-loop system.

reference trajectory is $T_{ref}/T_s = 4$. The parameters in the model for the controller on processor P_1 are

$$\begin{aligned}
\mathbf{nu_1'(k)} &= [\,u_1'(k) \quad u_2'(k) \quad u_3'(k)\,]^T \\
&= \left[u_1(k) \quad e^{-\frac{T_s}{T_{ref}}}u_2(k) + \left(1 - e^{-\frac{T_s}{T_{ref}}}\right)B_2 \quad e^{-\frac{T_s}{T_{ref}}}u_3(k) \right. \\
&\quad \left. + \left(1 - e^{-\frac{T_s}{T_{ref}}}\right)B_3\right]^T, \\
\mathbf{G_1(k)} &= \begin{bmatrix} g_1(k) & 0 & 0 \\ 0 & g_2(k) & 0 \\ 0 & 0 & g_3(k) \end{bmatrix}, \\
\mathbf{F_1} &= \begin{bmatrix} c_{11} & c_{21} & c_{31} & c_{42} & 0 & 0 & 0 & 0 \\ 0 & 0 & c_{32} & c_{41} & c_{51} & c_{62} & 0 & 0 \\ 0 & c_{22} & c_{33} & 0 & 0 & c_{61} & c_{71} & c_{83} \end{bmatrix}, \\
\mathbf{\Delta r_1(k)} &= [\,\Delta r_1(k) \quad \Delta r_2(k) \quad \Delta r_3(k) \quad \Delta r_4(k) \quad \Delta r_5(k) \\
&\quad \Delta r_6(k) \quad \Delta r_7(k) \quad \Delta r_8(k)\,]^T, \\
\mathbf{B_1} &= [\,B_1 \quad B_2 \quad B_3\,]^T.
\end{aligned}$$

The solution for the controller on P_1 is of the form

$$\begin{aligned}
\mathbf{\Delta nr_1^1(k)} = &\begin{bmatrix} k_{11}^1 & k_{12}^1 & k_{13}^1 \\ \vdots & \vdots & \vdots \\ k_{81}^1 & k_{82}^1 & k_{83}^1 \end{bmatrix} \mathbf{nu_1'(k)} \\
&+ \begin{bmatrix} h_{11}^1 & \cdots & h_{18}^1 \\ \vdots & \ddots & \vdots \\ h_{81}^1 & \cdots & h_{88}^1 \end{bmatrix} \mathbf{\Delta nr_1(k-1)} \\
&+ \begin{bmatrix} e_{11}^1 & e_{12}^1 & e_{13}^1 \\ \vdots & \vdots & \vdots \\ e_{81}^1 & e_{82}^1 & e_{83}^1 \end{bmatrix} \mathbf{B_1}.
\end{aligned} \tag{11}$$

The superscript 1 denotes that the solution is for the controller on P_1.

Following Step 2, we construct the feedback and feedforward matrices for (9). Since controller C_1 manipulates the control variables Δr_1, Δr_2, and Δr_3, the first three rows of the matrices $\mathbf{K}$ and $\mathbf{L}$ are constructed by the first three rows of $\mathbf{K_1}$. The first three rows of the matrix $\mathbf{E}$ are constructed by the first three rows of $\mathbf{E_1}$ and $\mathbf{K_1}$. The first three rows of the matrix $\mathbf{H}$ are constructed by the first three rows of the matrix $\mathbf{H_1}$. The matrices $\mathbf{K}$, $\mathbf{H}$, and $\mathbf{E}$ can be completed by the corresponding matrices from controllers on other processors. Then, we can derive the composite system (10).

If the $\mathbf{G(k)}$ is time invariant, the positions of the poles show the stability of the system. The poles are functions of the system gains in $\mathbf{G(k)}$. The closed-loop system has 31 poles. Our Matlab program allows us to analyze the system stability under any $\mathbf{G(k)}$. For example, Fig. 4 shows the root locus of the closed-loop system by DEUCON for the case that all nonzero elements of $\mathbf{G(k)}$ have the same value, denoted by g. The root locus is the trajectory of the poles of the closed-loop system as g varies. The dotted circle is the unit circle. It shows that all poles are within the unit circle for $0 < g < 2$. Furthermore, the DC gain of the closed-loop system is the identity matrix for $0 < g < 2$. Therefore, the system is stable. Our analysis proves that DEUCON can provide robust utilization guarantees to the example system even when actual execution times deviate significantly from the estimation. For instance, our results indicate that DEUCON can converge to the desired utilizations on all processors even if the execution time of every task is 90 percent lower $(g = 0.1)$ or 90 percent higher $(g = 1.9)$ than the estimation as long as the range of task rates are not violated. We validate this analysis through simulations presented in Section 6. When $\mathbf{G(k)}$ varies along time, the stability of the system for each individual value of $\mathbf{G}$ does not imply the stability of the system with varying $\mathbf{G}$. We need to identify a range of $\mathbf{G}$ for which there exists a common Lyapunov function for all $\mathbf{G}$s. Using a linear matrix inequality (LMI) technique [3], we can find a common Lyapunov function for systems with any $0 < g < 2$. Therefore, the system is stable if all nonzero elements of $\mathbf{G(k)}$ have the same value $g(k)$ and $g(k)$ varies between 0 and 2.

5.6 Discussions

DEUCON is particularly efficient for systems with long-running tasks with fixed routes. When the task allocation changes dynamically due to task arrival, termination, or route changes, any controller whose direct or indirect neighborhoods are affected needs to reconfigure its internal model and update its neighborhood information in order to maintain utilization control. For instance, when a task changes its route, all the controllers within its old and new neighborhoods need to reconfigure their models and

neighborhood information. Note that, thanks to the localized control scheme of DEUCON, usually only a subset of the controllers needs to perform reconfiguration in response a workload change. In an earlier work, we have successfully implemented this controller reconfiguration mechanism for a EUCON controller in a DRE middleware [34].

DEUCON is designed to control the aggregate utilization of each processor. An alternative to the approach is per-task control in which the system monitors and controls the utilization of each individual task. There exists a trade-off between the granularity and the overhead of control. Although a per-task control approach may enable fine-grained control over individual tasks and achieve performance isolation among tasks, it also introduces a higher overhead because it requires monitoring the utilization of every subtask, which is commonly implemented as a separate thread in real-time middleware systems. In contrast, although the per-processor control approach adopted by DEUCON cannot support fine-grained control over individual tasks, it can be more efficient because it only needs to monitor the utilization of each processor. In general, the choice of control approaches depends on application requirements and platform characteristics. We adopt the per-processor control approach because we aim to implement DEUCON as a middleware service for large-scale DRE systems. Monitoring the utilization of every subtask at the middleware (user) level may introduce a nonnegligible overhead in those systems.

6 Simulation Results

In this section, we first describe the simulation settings. We then compare the performances and overheads of DEUCON and EUCON. We choose EUCON as the baseline for performance as it is the only available utilization control algorithm for DRE systems with end-to-end tasks. Previous results showed that EUCON significantly outperformed a common open-loop approach that assigned fixed task rates based on estimated execution times [23]. Finally, we evaluate the scalability of DEUCON in large systems using randomly generated workloads.

6.1 Simulation Setup

Our simulation environment is composed of an event-driven simulator implemented in C++ and a set of controllers implemented in Matlab (R12). The simulator implements the utilization monitors, the rate modulators, and the DRE system with an interface to the controllers. The subtasks on each processor are scheduled by the Rate Monotonic Scheduling (RMS) algorithm [19]. The precedence constraints among subtasks are enforced by the release guard protocol [33]. The controllers are based on the *lsqlin* least squares solver in Matlab. The simulator opens a Matlab process and initializes all the controllers at start time. At the end of each sampling period, the simulator collects the local utilization, the predicted neighborhood utilizations, and the concerned task rates for each controller and then calls the controller in Matlab. The controllers compute the control input $\Delta r(k)$ and return it to the simulator. The simulator then calls the rate modulators on each processor to adjust the rates of its mastered tasks.

Each task has its end-to-end deadline as $d_i = n_i/r_i(k)$, 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, $1/r_i(k)$. The schedulable utilization bound of RMS [19] $B_i = m_i(2^{1/m_i} - 1)$ is used as the utilization setpoint on each processor, where m_i is the number of subtasks on P_i. All (sub)tasks meet their (sub)deadlines if the utilization setpoint on every processor is enforced.[3]

A medium-sized workload (as shown in Fig. 3) is used in our experiments. It includes 21 tasks (with a total of 40 subtasks) executing on 10 processors. There are 14 end-to-end tasks running on multiple processors and seven local tasks. The execution time of each subtask follows a uniform distribution between its best-case and worst-case execution times. Worst-case execution times are configured to be 22 percent to 100 percent longer than the corresponding best-case execution times. The controller parameters used for this workload include the prediction horizon as 2 and the control horizon as 1. The control period $T_s = 1{,}000$ time units. The time constant T_{ref}/T_s used in (5) is set as 4. Each subtask has 2.5 to 6.7 instances released in each sampling period. The minimum and maximum rates of each task are 1/10th and 20 times its initial rate, respectively. We use the wide rate ranges in order to stress-test the stability of DEUCON in the face of wide variations in subtask execution times. Specific parameters of tasks are not shown due to space limitations.

To evaluate the robustness of DEUCON when execution times deviate from the estimation, the 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 actual 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 actual execution times) may be kept constant or changed dynamically in a run. When all subtasks share the same constant etf, it equals the system gain on every processor in the model, that is, $etf = g_{ii}(1 \le i \le m)$. In the following, we use $inversed\, etf\ (ietf)$ defined by $ietf_{ij}(k) = 1/etf_{ij}(k)$ because we are more interested in the situation when execution times are overestimated (that is, $etf < 1$).[4]

6.2 System Performance

In this subsection, we present two sets of simulation experiments. The first one evaluates DEUCON's system performance when task execution times deviate from the estimation. The second experiment tests DEUCON's ability to provide robust utilization guarantees when task execution times vary dynamically at runtime.

3. Other utilization bounds [16] can be used by DEUCON when the subdeadlines of subtasks are not equal to their periods.

4. In general, as discussed in [23], algorithms based on model predictive control and distributed model predictive control cause oscillation when the execution times are underestimated (that is, $etf > 1$).

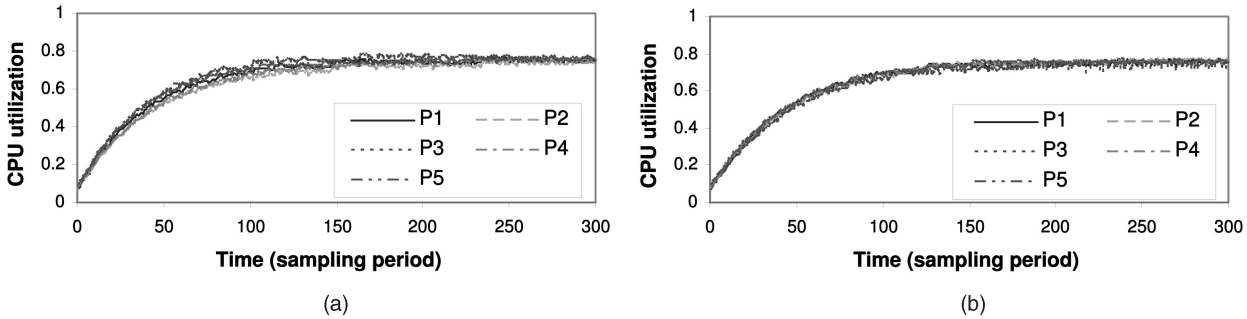

Fig. 5. CPU utilization of P_1 to P_5 $(\text{ietf} = 8)$. (a) DEUCON. (b) EUCON.

6.2.1 Steady Execution Times

In this experiment, all subtasks share a fixed execution-time factor $(ietf)$ in each run. Since it is commonly difficult to precisely estimate the execution times of real-time tasks in a DRE system, we stress-test DEUCON's performance when the real execution time significantly deviates from their estimations. Figs. 5a and 5b show the utilizations of processors P_1 to P_5 when execution times of tasks are *one-eighth* of their estimations. In this case, we can observe a noticeable difference in the transient state between DEUCON and EUCON. Although the utilizations of EUCON follow the same trajectory, utilizations of DEUCON diverge in the middle of the run and then converge to their setpoints in the end. Because DEUCON only uses neighborhood information to make local control decisions, the utilizations of different processors may follow different trajectories due to the different states in their respective neighborhoods. Despite this slight difference in the transient state, all utilizations converge to their setpoints within similar settling times. Both DEUCON and EUCON achieve the desired utilization guarantees in steady states.

To examine DEUCON's performance under different execution-time factors, we plot the mean and standard deviation of utilization on P_1 during each run in Fig. 6. Every data point is based on the measured utilization $u(k)$ from time $200T_s$ to $300T_s$ to exclude the transient response in the beginning of each run. Both EUCON and DEUCON achieve the desired utilizations for all tested execution-time factors within the $ietf$ range [0.5, 10]. In this range, the average utilizations under EUCON and DEUCON remain within ± 0.012 to the utilization setpoints and the standard deviations remain below 0.025. However, when $ietf = 8$, DEUCON's performance is slightly worse than that of EUCON, as its average utilization is 0.012 lower than its setpoint. In addition, EUCON has a high deviation when $ietf = 9$ because P_1 has a longer settling time under EUCON. As a result, the system is still in its transient state for part of the interval $[200T_s, 300T_s]$. We also observe that both EUCON and DEUCON suffer a standard deviation of ± 0.025 when $ietf = 0.5$. However, as a key benefit, both EUCON and DEUCON can achieve the desired utilizations even when execution times are severely overestimated. This capability is in sharp contrast to open-loop approaches, which are based on schedulability analysis. Open-loop approaches underutilize the processors in such cases.

To further investigate the CPU utilizations on other processors, Fig. 7 plots the average utilizations of all processors when $ietf$ is 5. The deviations of all utilizations are less than 0.008. We observe that, from P_2 to P_7, the differences between the utilizations and the setpoints for DEUCON are slightly larger than that of EUCON. However, all the differences are within the ± 0.009 range. In practice, such small steady-state errors can be handled by setting the setpoints to be slightly lower than the schedulable utilization bounds.

In summary, the simulation results demonstrate that DEUCON can achieve almost the same performance as EUCON for a wide range of $ietf$ ([0.5, 10] in our experiments). We also note that the range of $ietf$ corresponds to a system gain g in the range [0.1, 2]. Therefore, our simulation results validate the correctness of our stability analysis presented in Section 5.5.

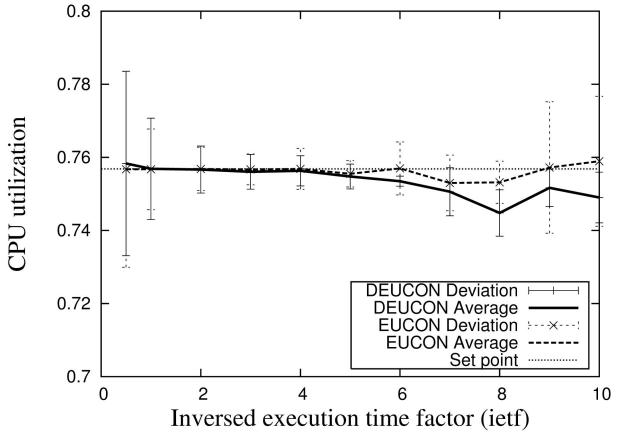

Fig. 6. The average and deviation of the CPU utilization of P_1 with different execution times.

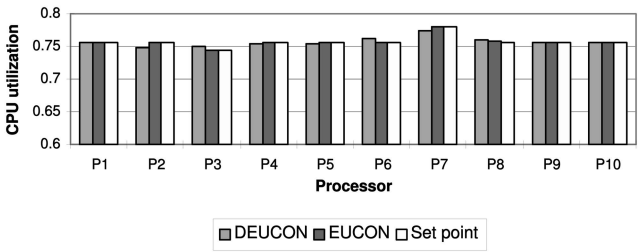

Fig. 7. Average CPU utilization $(\text{ietf} = 5)$.

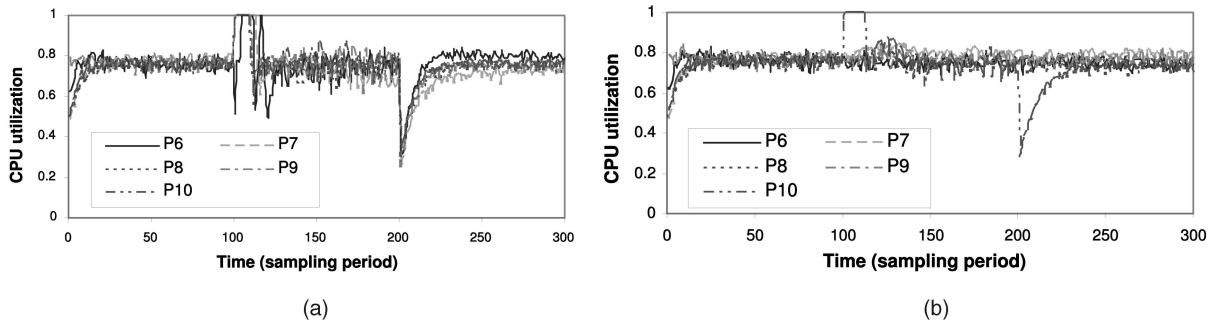

Fig. 8. CPU utilization of P_6 to P_{10} when execution times fluctuate at runtime. (a) Global fluctuation. (b) Local fluctuation on P_{10}.

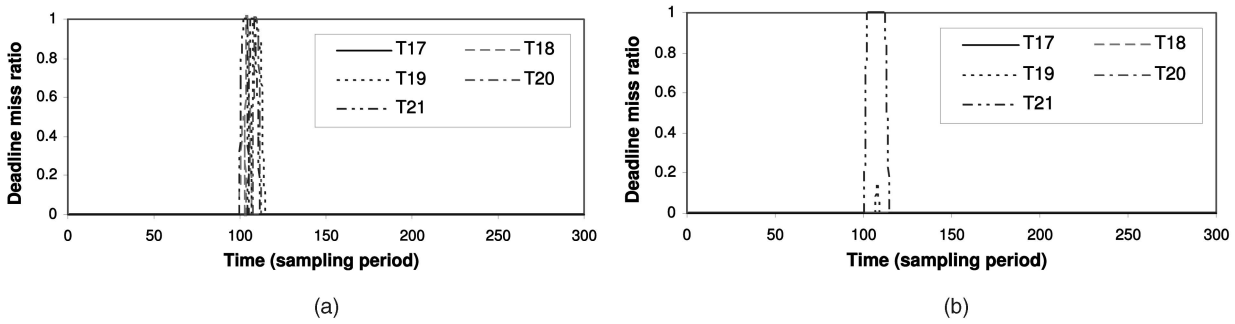

Fig. 9. Deadline miss ratio of T_{17} to T_{21} when execution times fluctuate at runtime. (a) Global fluctuation. (b) Local fluctuation on P_{10}.

6.2.2 Varying Execution Times

In this experiment, execution times vary *dynamically* at runtime. To investigate the robustness of DEUCON, we tested two scenarios of workload fluctuation. In the first set of runs, the average execution times on all processors change simultaneously. In the second set of runs, only the average execution times of the subtasks on P_{10} change dynamically, whereas those on the other processors remain unchanged. The first scenario represents a *global* load fluctuation and the second scenario represents a *local* load fluctuation on a part of the system.

Fig. 8a shows a typical run with global workload fluctuation. The $ietf$ is initially 1.0. At time $100T_s$, it is decreased to 0.56, which corresponds to a 79 percent increase in the execution times of all subtasks such that all processors are suddenly overloaded. Fig. 9a shows that the deadline miss ratios of tasks T_{17} to T_{21} increase suddenly from zero to almost 100 percent.[5] DEUCON responds to the overload by decreasing task rates that cause the utilizations on all processors to reconverge to their setpoints within $20T_s$. As a result, all end-to-end tasks meet their deadlines again. At time $200T_s$, the $ietf$ is increased to 1.67 corresponding to a 66 percent decrease in execution times. The utilizations on all processors drop sharply, causing DEUCON to dramatically increase task rates until the utilizations reconverge to their setpoints.[6] The system maintains stability and avoids any significant oscillation throughout the run, despite the variations in execution times.

In each run with local fluctuation, the $ietf$ on P_{10} follows the same variation as the global fluctuation, whereas all the other processors have a fixed $ietf$ of 1.0. As shown in Fig. 8b, the utilization of P_{10} converges to its setpoint after the significant variation of execution times at $120T_s$ and $250T_s$, respectively. We also observe that the other processors experience only a slight utilization fluctuation after the execution times change on P_{10}. This result demonstrates that DEUCON effectively handles the coupling among processors during rate adaptation. Fig. 9b shows that only tasks T_{19} and T_{21} have deadline misses shortly after the execution time increase on processor P_{10} because T_{19} and T_{21} are located on P_{10} and their task rates are lower than the task rates of T_{16} and T_{17}, which are the other two tasks on P_{10}, as shown in Fig. 3. As a result of RMS scheduling [19], T_{21} and T_{19} have the lowest priorities and are therefore most affected by processor overload. As the utilization of P_{10} reconverges to the setpoint, the deadline miss ratios of both tasks drop to zero. Our results demonstrate that DEUCON can effectively control the real-time performance of a DRE system in the face of load fluctuation at runtime.

6.3 Overhead

As discussed in Section 4, a major limitation of a centralized controller is that the runtime overhead is related to the size of the entire system. In contrast, the overhead of each local controller in DEUCON is just a function of its neighborhood size. Fig. 10 compares the size of the entire system with the neighborhood size of each processor for the medium-size workload. The centralized EUCON controller needs to model all 10 processors and 21 tasks in the system. In contrast, the average for DEUCON controllers is only

5. We choose to show the deadline miss ratios of tasks T_{17} to T_{21} because they are located on P_6 to P_{10} and three of them (T_{17}, T_{19}, and T_{21}) are located on P_{10}, which is chosen to show local fluctuation.

6. Only the results of P_6 to P_{10} are included in Fig. 8 for clarity. Performance of P_1 to P_5 are similar.

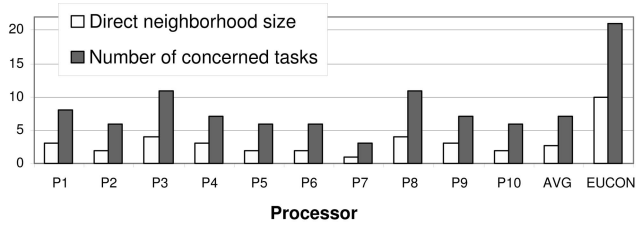

Fig. 10. Entire system size versus neighborhood size.

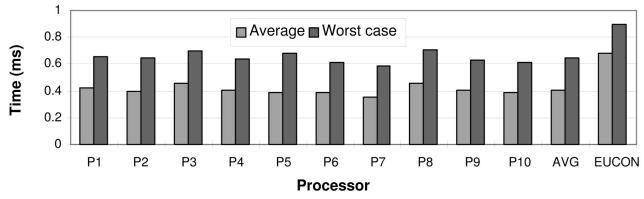

Fig. 11. Controller execution time in Matlab.

2.6 processors and 7.1 tasks, corresponding to a reduction by 74 percent and 66 percent, respectively.

To estimate the computation overhead of the controllers, we measure the execution time of the least squares solver that dominates the computation cost on a 2-GHz Pentium 4 PC with 256-MByte RAM. In order to minimize the effect of the time delay caused by the interprocess communication (IPC) between the simulator and the Matlab process, for each control computation invoked in a sampling period, we use a single Matlab command to run this least squares solver for 1,000 times as a subroutine. We then collect the average of the 1,000 runs through 300 sampling periods in a system run. The data shown in Fig. 11 is the average and the worst case of those 300 periods. The average of all controllers in DEUCON is only 59 percent and that of EUCON's centralized controller is 71 percent. We note that the speedup in execution times is not strictly polynomial in the numbers of neighbors and concerned tasks as one would expect from the theoretical complexity of MPC algorithms. This is attributed to the difference between the *actual average* execution time of Matlab's *lsqlin* solver and the *theoretical worst case* computational complexity. In addition, the initialization cost in the optimization calculations is not negligible for relatively small-scale problems in our workload.

We now investigate DEUCON's communication overhead. As mentioned in Section 5, a controller's communication overhead is a function of the number of processors communicating with it.[7] To estimate the communication overhead due to utilization exchange, we count the number of processors from which a controller receives the predicted utilizations. This is equal to the number of direct neighbors of the controller. To estimate the communication overhead due to task rate exchange, we count the processors from which a controller receives the actual rate changes for one or more of its concerned tasks. The set of processors communicating with a controller is the union of these two processor sets. In Fig. 12, we can see that DEUCON's average estimated per-controller communication overhead

7. Multiple data values (utilizations and/or rates) from the same processor can be easily combined to a single message in a real system implementation.

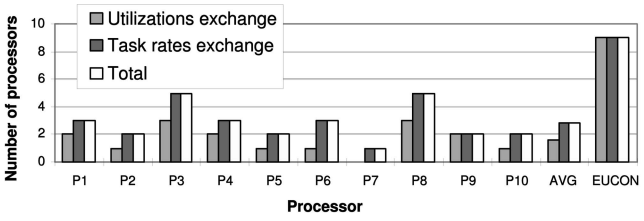

Fig. 12. Estimated communication overhead.

is 33 percent of the EUCON controller's communication overhead.

6.4 Scalability

Our final set of simulations evaluate the scalability of DEUCON in large systems. In all the following simulations, we employ randomly generated workloads. All subtasks are randomly allocated to processors such that every processor has the same number of subtasks. The number of subtasks per processor is fixed at five in all of the following simulations. To evaluate the scalability of DEUCON, we increase both the number of processors and the total number of subtasks in the systems proportionally.

Since the total number of subtasks is the product of the number of tasks and the number of subtasks per task, the system size can be varied in two ways:

- *Case 1*. We keep the number of subtasks per task fixed at five and increase both the number of tasks and the number of processors from 100 to 1,000.
- *Case 2*. We keep the number of tasks fixed at 500 and then increase the number of subtasks per task from one to 10 and the number of processors from 100 to 1,000.

Fig. 13 shows the direct neighborhood size, the number of concerned tasks, and the number of communicated processors of a controller in Case 1. Every result is the average or worst-case value of all controllers in the system. We can see that the size of the direct neighborhood remains almost constant despite the tenfold increase in the number of processors. At the same time, the number of concerned tasks and communicated processors increases very slowly. Even in the system with 1,000 processors, a controller only communicates with fewer than 34 processors on average. These results demonstrate that the per-controller overhead of DEUCON is almost independent of the total size of the system when the number of subtasks per task remains fixed.

We then investigate Case 2. Fig. 14 shows that the three overhead metrics increase when the numbers of processors and subtasks increase. This is because, when each task has more subtasks, the number of processors in the control model of a controller also increases, resulting in a larger neighborhood. However, our results show that the per-controller overhead remains moderate even when each task has a high number of subtasks. For example, a controller communicates with only 59.3 of 1,000 processors on average and 166 in the worst case even when each task has 10 subtasks. We note that, in practice, it is rare for a task to have an extremely large number of subtasks.

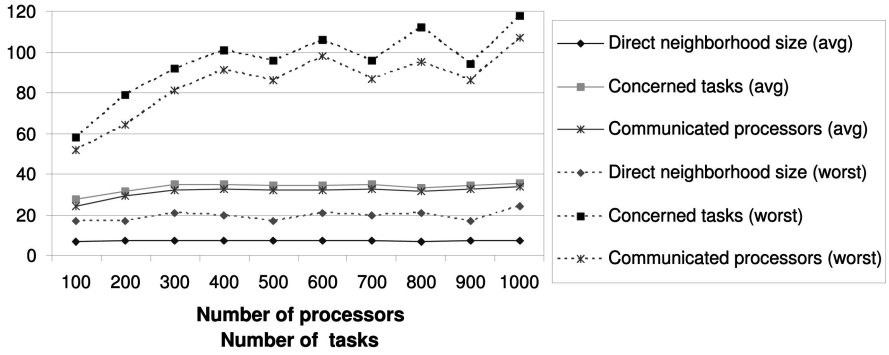

Fig. 13. Per-controller overhead when tasks increase with processors.

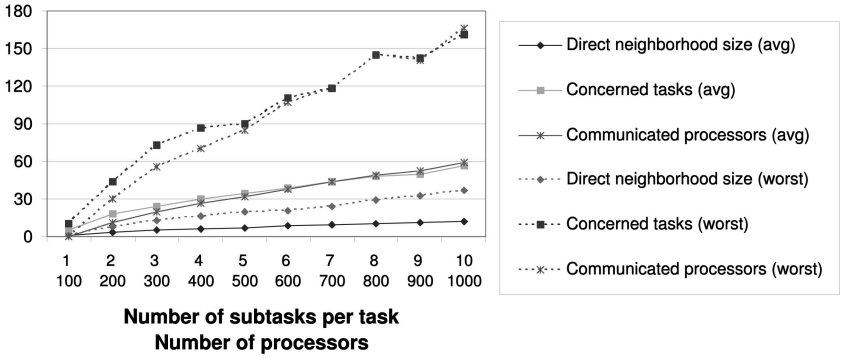

Fig. 14. Per-controller overhead when subtasks increase with processors.

Moreover, we observe that real-world systems may allocate subtasks in a clustered fashion; that is, all subtasks of a subsystem tend to share several processors and only a small number of tasks run across multiple subsystems. We expect such a clustered allocation to result in an even smaller neighborhood size than the random allocation in our simulations.

7 Conclusions

We have presented the DEUCON algorithm for dynamically controlling the utilization of DRE systems. DEUCON features a novel decentralized control structure to handle the coupling among multiple processors due to end-to-end tasks. Both stability analysis and simulation results demonstrate that DEUCON achieves robust utilization guarantees even when task execution times deviate significantly from the estimation or changes dynamically at runtime. Furthermore, DEUCON can significantly improve the system scalability by distributing the computation and communication cost from a central processor to local controllers distributed in the whole system and tolerating network delays.

Acknowledgments

This work was supported in part by a start-up grant from the University of Tennessee and a US National Science Foundation CAREER award (CNS-0448554). This work was done while Dong Jia was with Carnegie Mellon University, Pittsburgh, Pennsylvania.

References

[1] T. Abdelzaher, J. Stankovic, C. Lu, R. Zhang, and Y. Lu, "Feedback Performance Control in Software Services," *IEEE Control Systems*, vol. 23, no. 3, June 2003.

[2] L. Abeni, L. Palopoli, G. Lipari, and J. Walpole, "Analysis of a Reservation-Based Feedback Scheduler," *Proc. IEEE Real-Time Systems Symp. (RTSS '02)*, Dec. 2002.

[3] S. Boyd, L.E. Ghaoui, E. Feron, and V. Balakrishnan, "Linear Matrix Inequalities in System and Control Theory," *Soc. Industrial and Applied Math. (SIAM)*, 1994.

[4] S. Brandt, G. Nutt, T. Berk, and J. Mankovich, "A Dynamic Quality of Service Middleware Agent for Mediating Application Resource Usage," *Proc. IEEE Real-Time Systems Symp. (RTSS '98)*, Dec. 1998.

[5] G.C. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni, "Elastic Scheduling for Flexible Workload Management," *IEEE Trans. Computers*, vol. 51, no. 3, pp. 289-302, Mar. 2002.

[6] M. Caccamo, G. Buttazzo, and L. Sha, "Handling Execution Overruns in Hard Real-Time Control Systems," *IEEE Trans. Computers*, vol. 51, no. 7, pp. 835-849, July 2002.

[7] E.F. Camacho and C. Bordons, *Model Predictive Control*. Springer, 1999.

[8] E. Camponogara, D. Jia, B. Krogh, and S. Talukdar, "Distributed Model Predictive Control," *Control Systems Magazine*, vol. 22, no. 1, pp. 44-52, Feb. 2002.

[9] R. Carlson, "Sandia SCADA Program High-Security SCADA LDRD Final Report," Sandia Report SAND2002-0729, 2002.

[10] A. Cervin, J. Eker, B. Bernhardsson, and K.-E. Arzen, "Feedback-Feedforward Scheduling of Control Tasks," *Real-Time Systems*, vol. 23, no. 1, pp. 25-53, July 2002.

[11] J.D. Fernandez and A.E. Fernandez, "SCADA Systems: Vulnerabilities and Remediation," *J. Computing in Small Colleges*, vol. 20, no. 4, pp. 160-168, 2005.

[12] A. Goel, J. Walpole, and M. Shor, "Real-Rate Scheduling," *Proc. IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS '04)*, 2004.

[13] D. Henriksson and T. Olsson, "Maximizing the Use of Computational Resources in Multi-Camera Feedback Control," *Proc. IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS '04)*, May 2004.

[14] B. Kao and H. Garcia-Molina, "Deadline Assignment in a Distributed Soft Real-Time System," *IEEE Trans. Parallel and Distributed Systems,* vol. 8, no. 12, pp. 1268-1274, Dec. 1997.
[15] X. Koutsoukos, R. Tekumalla, B. Natarajan, and C. Lu, "Hybrid Supervisory Utilization Control of Real-Time Systems," *Proc. IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS '05),* 2005.
[16] J.P. Lehoczky, "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadline," *Proc. IEEE Real-Time Systems Symp. (RTSS '90),* 1990.
[17] F. Lewis and V. Syrmos, *Optimal Control,* second ed. John Wiley & Sons, 1995.
[18] S. Lin and G. Manimaran, "Double-Loop Feedback-Based Scheduling Approach for Distributed Real-Time Systems," *Proc. Int'l Conf. High Performance Computing (HiPC '03),* pp. 268-278, 2003.
[19] C. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *J. ACM,* vol. 20, no. 1, pp. 46-61, Jan. 1973.
[20] J.W.S. Liu, *Real-Time Systems.* Prentice Hall, 2000.
[21] C. Lu, J. Stankovic, G. Tao, and S. Son, "Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms," *Real-Time Systems,* vol. 23, nos. 1/2, pp. 85-126, July 2002.
[22] C. Lu, X. Wang, and C. Gill, "Feedback Control Real-Time Scheduling in ORB Middleware," *Proc. IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS '03),* May 2003.
[23] C. Lu, X. Wang, and X. Koutsoukos, "Feedback Utilization Control in Distributed Real-Time Systems with End-to-End Tasks," *IEEE Trans. Parallel and Distributed Systems,* vol. 16, no. 6, pp. 550-561, June 2005.
[24] J. Maciejowski, *Predictive Control with Constraints.* Prentice Hall, 2002.
[25] P. Marti, G. Fohler, P. Fuertes, and K. Ramamritham, "Improving Quality-of-Control Using Flexible Timing Constraints: Metric and Scheduling," *Proc. IEEE Real-Time Systems Symp. (RTSS '02),* 2002.
[26] M.D. Natale and J. Stankovic, "Dynamic End-to-End Guarantees in Distributed Real-Time Systems," *Proc. IEEE Real-Time Systems Symp. (RTSS '94),* 1994.
[27] R. Rajkumar, L. Sha, and J.P. Lehoczky, "Real-Time Synchronization Protocols for Multiprocessors," *Proc. IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS '88),* Dec. 1988.
[28] D. Seto, J.P. Lehoczky, L. Sha, and K.G. Shin, "On Task Schedulability in Real-Time Control System," *Proc. IEEE Real-Time Systems Symp. (RTSS '96),* Dec. 1996.
[29] V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, and Z. Lu, "Power-Aware QoS Management in Web Servers," *Proc. IEEE Real-Time Systems Symp. (RTSS '03),* 2003.
[30] C. Shen, K. Ramamritham, and J.A. Stankovic, "Resource Reclaiming in Multiprocessor Real-Time Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 4, no. 4, pp. 382-397, Apr. 1993.
[31] J.A. Stankovic, T. He, T. Abdelzaher, M. Marley, G. Tao, S. Son, and C. Lu, "Feedback Control Scheduling in Distributed Real-Time Systems," *Proc. IEEE Real-Time Systems Symp. (RTSS '01),* 2001.
[32] D.C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole, "A Feedback-Driven Proportion Allocator for Real-Rate Scheduling," *Operating Systems Design and Implementation,* pp. 145-158, 1999.
[33] J. Sun and J.W.-S. Liu, "Synchronization Protocols in Distributed Real-Time Systems," *Proc. IEEE Int'l Conf. Distributed Computing Systems (ICDCS '96),* 1996.
[34] X. Wang, C. Lu, and X. Koutsoukos, "Enhancing the Robustness of Distributed Real-Time Middleware via End-to-End Utilization Control," *Proc. IEEE Real-Time Systems Symp. (RTSS '05),* 2005.
[35] Y. Ye, *Interior Point Algorithms: Theory and Analysis.* John Wiley & Sons, 1997.
[36] F. Zhao, X.D. Koutsoukos, H.W. Haussecker, J. Reich, P. Cheung, and C. Picardi, "Distributed Monitoring of Hybrid Systems: A Model-Directed Approach," *Proc. Int'l Joint Conf. Artificial Intelligence (IJCAI '01),* pp. 557-564, 2001.
[37] Y. Zhu and F. Mueller, "Feedback EDF Scheduling Exploiting Dynamic Voltage Scaling," *Proc. IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS '04),* 2004.

Xiaorui Wang received the BS degree from Southeast University, China, in 1995, the MS degree from the University of Louisville in 2002, and the PhD degree from Washington University in St. Louis in 2006, all in computer science. He is an assistant professor in the Department of Electrical and Computer Engineering at the University of Tennessee, Knoxville. In 2005, he worked at the IBM Austin Research Laboratory, designing power control algorithms for IBM high-performance computing servers. From 1998 to 2001, he was a senior software engineer and then a project manager at Huawei Technologies, China, developing distributed management systems for optical networks. His research interests include real-time embedded systems, distributed systems, power-aware computing, and wireless sensor networks. He is a member of the IEEE and the IEEE Computer Society.

Dong Jia received the BS and MS degrees in control theory and application from Xi'an Jiaotong University, Xi'an, Shaanxi, China, in 1996 and 1999, respectively, and the PhD degree in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, in 2003. He is now working in the Control Design Automation group at The Mathworks. His current research interests include distributed control and signal processing, modeling and simulation of large-scale systems, code generation for embedded systems.

Chenyang Lu received the BS degree from the University of Science and Technology of China, Hefei, China, in 1995, the MS degree from the Chinese Academy of Sciences, Beijing, in 1997, and the PhD degree from the University of Virginia, Charlottesville, in 2001, all in computer science. He is an assistant professor in the Department of Computer Science and Engineering at Washington University in St. Louis. He is an author and coauthor of more than 60 refereed publications. His current research interests include distributed real-time embedded systems and middleware, wireless sensor networks, and adaptive quality-of-service control. He received a US National Science Foundation Faculty Early Career Development (CAREER) Award (2005) and a Best Paper Award at the International Conference on Distributed Computing in Sensor Systems (DCOSS '06). He is a member of the IEEE and the IEEE Computer Society.

Xenofon Koutsoukos (S'95, M'00) received the diploma in electrical and computer engineering from the National Technical University of Athens (NTUA), Greece, in 1993, the MS degrees in electrical engineering and applied mathematics in 1998, and the PhD degree in electrical engineering from the University of Notre Dame in 2000. From 2000 to 2002, he was a member of the research staff in the Xerox Palo Alto Research Center (PARC) working in the Embedded Collaborative Computing Area. Since 2002, he has been with the Department of Electrical Engineering and Computer Science at Vanderbilt University, where he is currently an assistant professor and a senior research scientist in the Institute for Software Integrated Systems (ISIS). His research interests are in the areas of hybrid systems, real-time embedded systems, and sensor networks. He has authored or coauthored more than 60 technical papers and is a coinventor of three US patents. He is a recipient of the US National Science Foundation Faculty Early Career Development (CAREER) Award in 2004. He is a member of the IEEE and the IEEE Computer Society.