Report Number: WUCSE-2005-44

2005-09-23

Decentralized Utilization Control in Distributed Real-Time Systems

Xiaorui Wang, Dong Jia, Chenyang Lu, and Xenofon Koutsoukos

Many real-time systems must control their CPU utiliza-tions in order to meet end-to-end deadlines and prevent over-load. Utilization control is particularly challenging in dis-tributed real-time systems with highly unpredictable work-loads and a large number of end-to-end tasks and processors. This paper presents the Decentralized End-to-end Utilization CONtrol (DEUCON) algorithm that can dynamically enforce desired utilizations on multiple processors in such systems. In contrast to centralized control schemes adopted in earlier work, DEUCON features a novel decentralized control struc-ture that only requires localized coordination among neigh-bor processors. DEUCON is systematically designed based on recent advances in distributed model predictive control... **Read complete abstract on page 2.**

Decentralized Utilization Control in Distributed Real-Time Systems

Xiaorui Wang, Dong Jia, Chenyang Lu, and Xenofon Koutsoukos

Complete Abstract:

Many real-time systems must control their CPU utiliza-tions in order to meet end-to-end deadlines and prevent over-load. Utilization control is particularly challenging in dis-tributed real-time systems with highly unpredictable work-loads and a large number of end-to-end tasks and processors. This paper presents the Decentralized End-to-end Utilization CONtrol (DEUCON) algorithm that can dynamically enforce desired utilizations on multiple processors in such systems. In contrast to centralized control schemes adopted in earlier work, DEUCON features a novel decentralized control struc-ture that only requires localized coordination among neigh-bor 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 dif-ferent processors and tolerate considerable communication delay between local controllers. Our results indicate that DEUCON can provide scalable and robust utilization con-trol for large-scale distributed real-time systems executing in unpredictable environments.

2005-44

Decentralized Utilization Control in Distributed Real-Time Systems

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

September 23, 2005

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 that can dynamically enforce desired utilizations on multiple processors in such systems. In contrast to centralized control schemes adopted in earlier work, DEUCON features a novel decentralized control structure that only requires 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 scalable and robust utilization control for large-scale distributed real-time systems executing in unpredictable environments.

Decentralized Utilization Control in Distributed Real-Time Systems

Xiaorui Wang[†], Dong Jia[‡], Chenyang Lu[†], Xenofon Koutsoukos[§]
[†]*Department of Computer Science and Engineering, Washington University in St. Louis, St. Louis, MO 63130*
[‡]*Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213*
[§] *Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN 37235*
{*wang, lu*}*@cse.wustl.edu djia@andrew.cmu.edu xenofon.koutsoukos@vanderbilt.edu*

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 that can dynamically enforce desired utilizations on multiple processors in such systems. In contrast to centralized control schemes adopted in earlier work, DEUCON features a novel decentralized control structure that only requires 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 scalable and robust utilization control for large-scale distributed real-time systems executing in unpredictable environments.

1 Introduction

Recent years have seen rapid growth of Distributed Real-time Embedded (DRE) applications executing in *unpredictable* environments in which workloads are unknown and vary significantly at run-time. 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 [11]. Likewise, the supervisory control and data acquisition (SCADA) systems for power grid control may experience dramatic load increase during a cascade power failure [8]. Furthermore, as DRE systems become connected to the Internet, they are exposed to load disturbances due to variable user requests and even cyber attacks [8]. As such systems become increasingly important to our society, a new paradigm of real-time computing based on *Adaptive QoS Control (AQC)* has received significant attention. In contrast to traditional approaches to real-time systems that rely on accurate knowledge about 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 an important instance of AQC called *utilization control* for distributed soft real-time systems. The goal of utilization control is to enforce 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.

DRE systems introduce many new research challenges that have not been addressed in earlier work on single-processor systems. First, they require *multi-input-multi-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* [17], where a task may comprise of a chain of subtasks on different processors. In such systems, the CPU utilizations of different processors cannot be controlled independently from others. 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 (e.g. 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 [20], 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 four-fold.

- We propose a new approach for decomposing the global multi-processor 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 desired utilizations of multiple processors through localized coordination among controllers.
- We give control analysis based on the *distributed model predictive control* (DMPC) theory [7] which establishes the stability properties of the DEUCON algorithm in face of uncertain task execution times.
- We present simulation results showing that DEUCON can provide robust statistical 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.

2 Related Work

Traditional approaches for handling end-to-end tasks such as end-to-end scheduling [29] and distributed priority ceiling [23] rely on schedulability analysis, which requires *a priori* knowledge about worst-case execution times. When task execution times are highly unpredictable, such open-loop approaches may severely under-utilize the system. An approach for dealing with unpredictable task execution times is resource reclaiming [5][26]. 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 COTS platforms [19].

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]. Several 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 [10] [28] that guarantee desired progress rates for real-time applications. Abeni et al. presented control analysis of a reservation-based feedback scheduler [2]. Authors of [18] developed feedback control scheduling algorithms that controlled the CPU utilization and deadline miss ratio. These algorithms have been implemented as a middleware service called FCS/nORB [19]. Feedback control scheduling has also been successfully applied to processor power control [32] and digital control applications [9] [25]. 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 which are not applicable to DRE systems with multiple processors.

Two recent papers [27][15] proposed feedback control scheduling algorithms for distributed real-time systems with *independent* tasks. These algorithms do not address the dependencies among processors caused by end-to-end tasks commonly available in DRE systems. Our earlier work produced EUCON (End-to-end Utilization CONtrol) [20] that is the first utilization control algorithm designed for DRE systems with end-to-end tasks. This control algorithm has also been validated and extended in a real middleware system [30]. EUCON manages and coordinates the adaptation of multiple processors with a *centralized* controller that cannot scale effectively in large-scale DRE systems because its communication and computation overhead depends on the size of an *entire* DRE system. We discuss EUCON in more detail in Section 5.

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 [17] implemented by many DRE applications. A 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 chain of subtasks $\{T_{ij}|1 \leq j \leq n_i\}$ located on different processors. The release of subtasks is subject to precedence constraints, i.e., subtask $T_{ij}(1 < j \leq n_i)$ cannot be released for execution

[1]Other control strategies such as task migration, quality level adaptation and possible combinations of them are subjects of our future research.

until its predecessor subtask T_{ij-1} is completed. If a non-greedy synchronization protocol (e.g., release guard [29]) is used to enforce the precedence constraints, all the subtasks of a periodic task share the same rate as the first subtask. Therefore, the rate of a task (and all its subtasks) can be adjusted by changing the rate of its first subtask. In this paper, the processor P_j hosting the first subtask of a task T_i is called T_i's *master processor* and we say P_j *masters* T_i. Only a task's master processor can change its rate.

Our task model has two important properties. First, while 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 run time. 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 (e.g., digital control [21][24], sensor update, and multimedia [3][4]) 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.

We assume that each task T_i has a *soft* end-to-end deadline related to its period. In an end-to-end scheduling approach [29], the deadline of an end-to-end task is divided into subdeadlines of its subtasks [12][22]. 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 its utilization remains below its schedulable utilization bound [13][16].

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 k^{th} sampling period, i.e., 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 set point on P_i. $r_j(k)$ is the invocation rate of task T_j in the $(k+1)^{th}$ sampling period.

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

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

subject to constraints

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

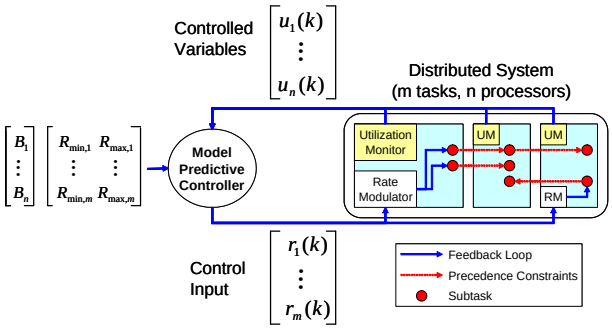


Figure 1. EUCON's feedback control loop with a centralized controller

The rate constraints ensure 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 set point as allowed by the constraints. The design goal is to ensure that all processors quickly converge to their utilization set points 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 set point 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 set points of all the processors in a DRE system.

4 EUCON: A Centralized Algorithm

In this section, we briefly describe the EUCON algorithm [20], which provides a starting point and baseline for our work.

As shown in Figure 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, $\mathbf{u(k)} = [u_1(k)...u_n(k)]^T$. 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 $\Delta r_i(k) = r_i(k) - r_i(k-1)$ $(1 \le i \le m)$.

The feedback control loop works as follows: (1) the utilization monitor on each processor P_i sends its utilization $u_i(k)$ 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 (i.e., processors that master at least one task); and (3) the rate modulators on master processors change the rates of tasks according to $\mathbf{r(k)}$. The details of the controller design in EUCON are described in [20].

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 run-time overhead depends on the size of an entire DRE system. Specifically, the worst-case computational complexity of a model predictive controller 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 (e.g., 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 desired utilizations without global information. In this section, we present the design of DEUCON based on a distributed model predictive control (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 [20], a DRE system can be approximated by the following *global* system model:

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

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 $f_{ij} = c_{jl}$ if a subtask T_{jl} of task T_j is allocated to processor P_i, and $f_{ij} = 0$ if no subtask of task T_j is allocated to processor P_i. $\mathbf{F}$ captures the *coupling* among processors due to end-to-end tasks. $\mathbf{G} = diag[g_1 \dots g_n]$ where g_i represents the ratio between the change in the actual utilization and its estimation. The exact value of g_i is *unknown* due to the unpredictability in execution times. Note that $\mathbf{G}$ describes the effect of uncertainty in workload on the utilization of a DRE system. As an example, Figure 2 shows a DRE system with five processors and five tasks. It is modeled by (2) with the following parameters:

$$\mathbf{u(k)} = \begin{bmatrix} u_1(k) \\ u_2(k) \\ u_3(k) \\ u_4(k) \\ u_5(k) \end{bmatrix}, \mathbf{G} = \begin{bmatrix} g_1 & 0 & 0 & 0 & 0 \\ 0 & g_2 & 0 & 0 & 0 \\ 0 & 0 & g_3 & 0 & 0 \\ 0 & 0 & 0 & g_4 & 0 \\ 0 & 0 & 0 & 0 & g_5 \end{bmatrix},$$

$$\mathbf{F} = \begin{bmatrix} c_{11} & 0 & 0 & 0 & c_{51} \\ c_{12} & c_{22} & 0 & 0 & 0 \\ 0 & c_{21} & c_{31} & 0 & 0 \\ 0 & 0 & c_{32} & c_{41} & 0 \\ 0 & 0 & c_{33} & c_{42} & 0 \end{bmatrix},$$

$$\mathbf{\Delta r(k)} = \begin{bmatrix} \Delta r_1(k) \\ \Delta r_2(k) \\ \Delta r_3(k) \\ \Delta r_4(k) \\ \Delta r_5(k) \end{bmatrix}$$

5.2 Problem Decomposition

Although our previous work showed that the above global system model is sufficient for designing a centralized controller for EUCON [20], 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.

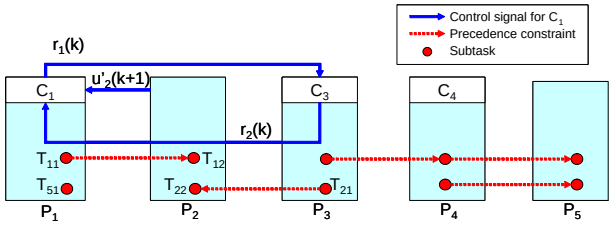


Figure 2. Data exchange between C_1 and its neighbors (other data exchanges are not shown)

Definition 2: The *concerned tasks* of P_i are the tasks which 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 Figure 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 us set $NR_i(k)$ includes 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:

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

subject to

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

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

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

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. $\mathbf{G_i}$ and $\mathbf{F_i}$ are defined in the same way as $\mathbf{G}$ 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 Figure 2 is modeled with the following parameters.

$$\mathbf{nu_1(k)} = \begin{bmatrix} u_1(k) \\ u_2(k) \end{bmatrix}, \mathbf{G_1} = \begin{bmatrix} g_1 & 0 \\ 0 & g_2 \end{bmatrix}$$

$$\mathbf{F_1} = \begin{bmatrix} c_{11} & 0 & c_{51} \\ c_{12} & c_{22} & 0 \end{bmatrix}, \mathbf{\Delta nr_1(k)} = \begin{bmatrix} \Delta r_1(k) \\ \Delta r_2(k) \\ \Delta r_5(k) \end{bmatrix}$$

From (4), C_1's local model is

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

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 which 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 which 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 depends 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 set point 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.

DEUCON is also a valid way to avoid single point of failure. Once a controller fails due to the failure of its host processor, all tasks on the host processor are immediately migrated to other proper processors in the system. The local controllers on these processors automatically adapt their control models to effectively manage the migrated tasks. As a result, the system's fault-tolerance capability is improved.

5.4 Controller Design

DEUCON employs a local controller on each *master* processor. Non-master processors do not need controllers because they cannot change the rate of any task. For the example shown in Figure 2, processors P_1, P_3 and P_4 each have a controller, while 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 [6] 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, e.g., $\mathbf{\Delta nr_i(k), \Delta nr_i(k+1|k), \ldots \Delta nr_i(k+M-1|k)}$, that minimizes the following cost function under the rate constraints.

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

where P is the *prediction horizon*, and M is the *control horizon*. The first term in the cost function represents the *tracking error*, i.e., 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 set points 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 set points. 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:

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

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 $\mathbf{G_i}$ in system model (4) are unknown in unpredicted environments, our controller assumes $\mathbf{G_i} = diag[1 \ldots 1]$, i.e., 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 desired utilization set points as long as $\mathbf{G_i}$ 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 online feedbacks used in our distributed model predictive control algorithm.

The controller computes the input trajectory $\mathbf{\Delta nr_i(k), \Delta nr_i(k+1|k), \ldots \Delta nr_i(k+M-1|k)}$ that minimizes the cost function subject to the rate constraints. This constrained optimization problem can be transformed to a standard constrained least square problem. Controller C_i can then use a standard least-square solver to solve this problem on-line. 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-square optimization is a convex nonlinear optimization, for which interior point methods require $O(n)$ Newton iterations [31], 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 $\mathbf{\Delta nr_i(k)}$ is applied as the control input and sent to the rate modulators. At 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.

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 set points $\mathbf{B}$, that is, $\lim_{\mathbf{k}\to\infty} \mathbf{u}(\mathbf{k}) = \mathbf{B}$. In this subsection we present stability analysis that allows users to analytically assess the robustness of DEUCON for their system with a range of uncertainties in term of task execution times. To ensure that the system can be stabilized, the constrained optimization problem must be 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). 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 [14], the local control decision is a linear function of the current value and the set points of the utilization 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 feed-forward matrices for each local controller i by solving its local control input $\mathbf{\Delta nr_i}$ based on the local system model (4) and reference trajectory (5). The solution is in the following form:

$$\mathbf{\Delta nr_i}(\mathbf{k}) = \mathbf{K_i nu_i}(\mathbf{k}) + \mathbf{H_i \Delta nr_i}(\mathbf{k-1}) + \mathbf{E_i B_i} \quad (8)$$

2. Construct the feedback and feed-forward matrices for the whole system (2) based on those for local system models derived in Step 1.

$$\mathbf{\Delta r}(\mathbf{k}) = \mathbf{K u}(\mathbf{k}) + \mathbf{H \Delta r}(\mathbf{k-1}) + \mathbf{EB} \quad (9)$$

   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 form

$$\begin{bmatrix} \mathbf{u}(\mathbf{k+1}) \\ \mathbf{\Delta r}(\mathbf{k}) \end{bmatrix} = \begin{bmatrix} \mathbf{I+GFK} & \mathbf{GFH} \\ \mathbf{K} & \mathbf{H} \end{bmatrix} \begin{bmatrix} \mathbf{u}(\mathbf{k}) \\ \mathbf{\Delta r}(\mathbf{k-1}) \end{bmatrix} + \begin{bmatrix} \mathbf{GFE} \\ \mathbf{E} \end{bmatrix} \mathbf{B} \quad (10)$$

   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 the control theory, if all poles locate inside the unit circle in the complex space and the DC gain matrix from the control to the state is the identity matrix, the state of the system, i.e., the processor utilizations, will converge to the set point.

The details of the above steps are not shown due to space limitations. We have developed a MATLAB program to perform the above stability analysis procedure automatically.

Example We now apply the stability analysis approach to the example system described in Figure 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 reference trajectory is $T_{ref}/T_s = 4$. The weights on all terms are 1. The parameters in the model for the controller on Processor P_1 are

$$\begin{aligned}
\mathbf{nu_1}(\mathbf{k}) &= \begin{bmatrix} u_1(k) & u_2(k) & u_3(k) \end{bmatrix}^T \\
\mathbf{G_1} &= \begin{bmatrix} g_1 & 0 & 0 \\ 0 & g_2 & 0 \\ 0 & 0 & g_3 \end{bmatrix} \\
\mathbf{F_1} &= \begin{bmatrix} c_{11} & c_{21} & c_{31} & c_{42} & 0 & 0 & 0 & 0 \\ 0 & 0 & c_{32} & c_{41} & c_{51} & c_{62} & 0 & 0 \\ 0 & c_{22} & c_{33} & 0 & 0 & c_{61} & c_{71} & c_{83} \end{bmatrix} \\
\mathbf{\Delta r_1}(\mathbf{k}) &= [\Delta r_1(k) \quad \Delta r_2(k) \quad \Delta r_3(k) \quad \Delta r_4(k) \\
&\quad\ \Delta r_5(k) \quad \Delta r_6(k) \quad \Delta r_7(k) \quad \Delta r_8(k)]^T \\
\mathbf{B_1} &= \begin{bmatrix} B_1 & B_2 & B_3 \end{bmatrix}^T
\end{aligned}$$

The solution for the controller on P_1 is of the form

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

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

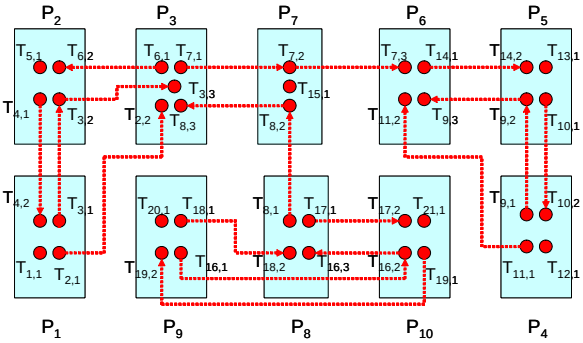


Figure 3. A medium size workload

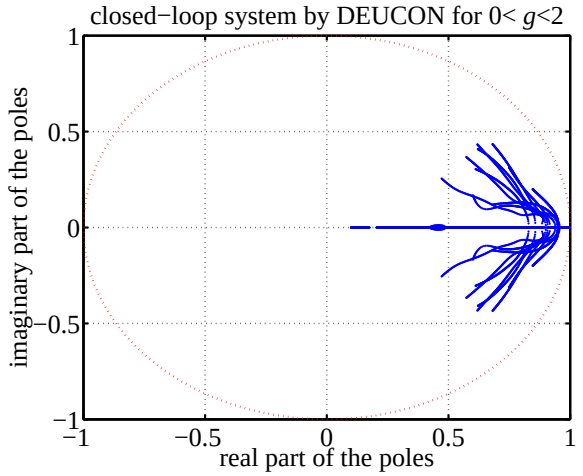


Figure 4. The root locus of the closed-loop system

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 matrix $\mathbf{K}$ is constructed by the first three rows of $\mathbf{K_1}$ as

$$\begin{bmatrix} k_{11}^1 & k_{12}^1 & k_{13}^1 & 0 & \cdots & 0 \\ k_{21}^1 & k_{22}^1 & k_{23}^1 & 0 & \cdots & 0 \\ k_{31}^1 & k_{32}^1 & k_{33}^1 & 0 & \cdots & 0 \end{bmatrix}.$$

Similarly, the first three rows of the matrices $\mathbf{H}$ and $\mathbf{E}$ are constructed as

$$\begin{bmatrix} h_{11}^1 & \cdots & h_{18}^1 & 0 & \cdots & 0 \\ h_{21}^1 & \cdots & h_{28}^1 & 0 & \cdots & 0 \\ h_{31}^1 & \cdots & h_{38}^1 & 0 & \cdots & 0 \end{bmatrix}$$

and

$$\begin{bmatrix} e_{11}^1 & e_{12}^1 & e_{13}^1 & 0 & \cdots & 0 \\ e_{21}^1 & e_{22}^1 & e_{23}^1 & 0 & \cdots & 0 \\ e_{31}^1 & e_{32}^1 & e_{33}^1 & 0 & \cdots & 0 \end{bmatrix}.$$

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).

The poles are functions of the system gains in $\mathbf{G}$. The closed-loop system has 31 poles. Our MATLAB program allows us to analyze the system stability under any $\mathbf{G}$. For example, Figure 4 shows the root locus of the closed-loop system by DEUCON for the case that all non-zero elements of $\mathbf{G}$ have the same value, denoted by g. 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 $0 < g < 2$. 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% lower (g = 0.1) or 90% 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.

6 Simulation Results

In this section, we first describe the simulation settings. We then compare the performance and overhead 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 [20].

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 distributed real-time system with an interface to the controllers. The subtasks on each processor are scheduled by the Rate Monotonic Scheduling (RMS) algorithm [16]. The precedence constraints among subtasks are enforced by the release guard protocol [29]. The controllers are based on the *lsqlin* least square solver in MATLAB. The simulator opens a MATLAB process and initializes all the controllers at start time. In 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'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, $1/r_i(k)$. The schedulable utilization bound of RMS [16],

$B_i = m_i(2^{1/m_i} - 1)$ is used as the utilization set point on each processor, 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[2].

A medium size workload (as shown in Figure 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 7 local tasks. The controller parameters used for this workload include the prediction horizon as 2 and the control horizon as 1. The control period $T_s = 1000$ time units. The time constant T_{ref} used in (5) is set as 4. 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 a same constant *etf*, it equals to the system gain on every processor in the model, i.e., $etf = g_{ii}(1 \leq i \leq m)$. In the following we use the *inversed etf* (*ietf*) because we are more interested in the situation when execution times are overestimated (i.e. $etf < 1$)[3]. Specifically, $ietf_{ij}(k) = 1/etf_{ij}(k)$.

6.2 System Performance

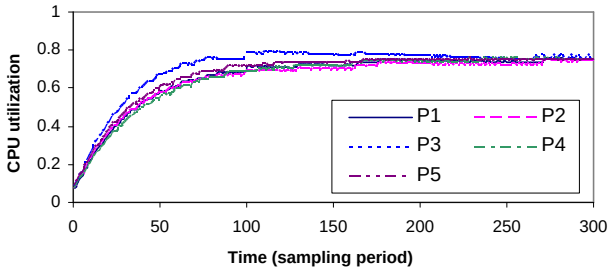
In this subsection we present two sets of simulation experiments. The first one evaluates DEUCON's system performance when task execution times deviate from the estimation. The second experiment tests DEUCON's ability to provide robust utilization guarantees when task execution times vary dynamically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

6.2.1 Steady Execution Times

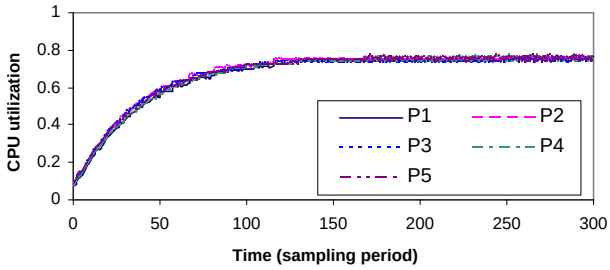
In this experiment, all subtasks share a fixed execution-time factor (*ietf*) in each run. Since it is commonly difficult to precisely estimate the execution times of real-time tasks in DRE system, we stress-test DEUCON's performance when real execution time significantly deviate from their estimations. Figures 5(a) and (b) 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. While the utilizations of EUCON follow the same trajectory, utilizations of DEUCON diverge in the middle of

[2]Other utilization bounds [13] can be used by DEUCON when the sub-deadlines of subtasks are not equal to their periods

[3]In general, as discussed in [20], algorithms based on model predictive control and distributed model predictive control cause oscillation when the execution times are underestimated (i.e. $etf > 1$).



(a) DEUCON



(b) EUCON

Figure 5. CPU utilization of P_1 to P_5 (ietf=8)

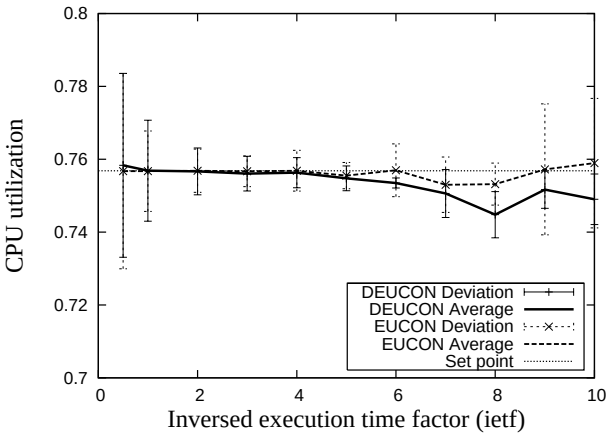


Figure 6. The average and deviation of the CPU utilization of P_1 with different execution times

the run and then converge to their set points in the end. The reason for this divergence is that each controller in DEUCON only utilizes local information and makes local decision. Despite this slight difference in the transient state, all utilizations converge to their set points within similar settling times. Both DEUCON and EUCON achieve 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 Figure 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 desired utilizations for all tested execution-time factors within the $ietf$ range $[0.5, 10]$. In this range, the aver-

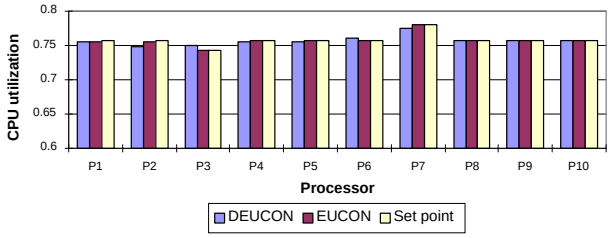


Figure 7. Average CPU utilization (ietf=5)

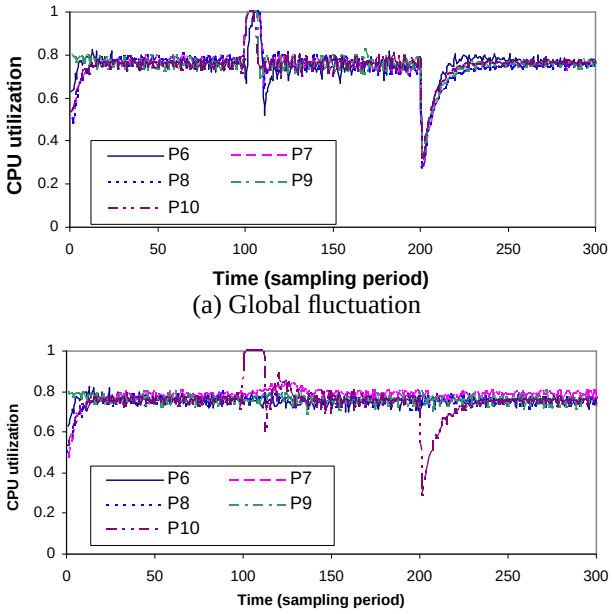
age utilizations under EUCON and DEUCON remain within ± 0.012 to the utilization set points and the standard deviations remain below 0.025. However, when $ietf = 8$, DEUCON's performance is slightly worse than that of EUCON, as its average utilization is 0.012 lower than its set point. In addition, EUCON 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 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 underutilizes the processors in such cases.

To further investigate the CPU utilizations on other processors, Figure 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 on P_2 to P_7, the difference between the utilizations and the set points 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 set points to 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 a range $[0.1, 2]$. Therefore, our simulation results validate the correctness of our stability analysis presented in Section 5.5.

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 execution times on P_{10} change dynamically, while those on the other processors remain unchanged. The first scenario represents *global* load fluctuation, while the second scenario represents *local* fluctuation on a part of the system.



(a) Global fluctuation

(b) Local fluctuation on P_{10}

Figure 8. CPU utilization of P_6 to P_{10} when execution times fluctuate at run-time

Figure 8(a) 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 an 79% increase in the execution times of all subtasks such that all processors are suddenly overloaded. DEUCON responds to the overload by decreasing task rates which causes the utilizations on all processors to re-converge to their set points within $20T_s$. At time $200T_s$, the *ietf* is increased to 1.67 corresponding to a 66% decrease in execution times. The utilizations on all processors drop sharply, causing DEUCON to dramatically increase task rates until the utilizations re-converge to their set points[4]. The system maintains stability and avoids any significant oscillation throughout the run, despite the variations in execution times.

In each run with local workload fluctuation, the *ietf* on P_{10} follows the same variation as the global fluctuation, while all the other processors have a fixed *ietf* of 1.0. As shown in Figure 8(b), the utilization of P_{10} converges to its set point after the significant variation of execution times at $120T_s$ and $250T_s$, respectively. We also observe that the other processors experience only 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. The performance results of DEUCON in this experiment are very close to EUCON's per-

[4]Only the results of P_6 to P_{10} are included in Figure 8 for clarity. Performance of P_1 to P_5 are similar.

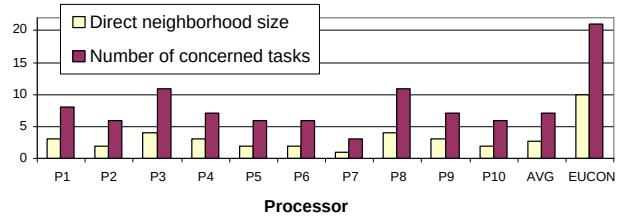


Figure 9. Entire system size vs. neighborhood size

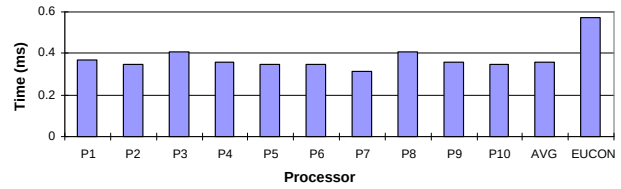


Figure 10. Controller execution time in MATLAB

formance reported in [20].

6.3 Overhead

As discussed in Section 4, a major limitation of a centralized controller is that the run-time 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. Figure 9 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 the 10 processors and the 21 tasks in the system. In contrast, the average for DEUCON controllers is only 2.6 processors and 7.1 tasks, corresponding to a reduction by 74% and 66%, respectively.

To estimate the *average* computation overhead of the controllers, we measure the execution time of the least square solver which dominates the computation cost on a 2GHz Pentium IV PC with 256MB RAM. In order to minimize the effect of the time delay caused by the IPC communication between the simulator and the MATLAB process, we use a single MATLAB command to run this least square solver for 1000 times as a subroutine. The data shown in Figure 10 is the average of those 1000 runs. The average execution time of all controllers in DEUCON is only 62% of EUCON's centralized controller. 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 difference between the *average* execution time of MATLAB's *lsqlin* solver and the *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[5]. To estimate communication overhead due to utilizations exchange, we count the number of processors from which a controller receives predicted utilizations. This is equal to the number of direct neighbors of the controller. To estimate communication overhead due to task rates 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. From Figure 11 we can see that DEUCON's average estimated per-controller communication overhead is 33% of the EUCON controller's communication overhead.

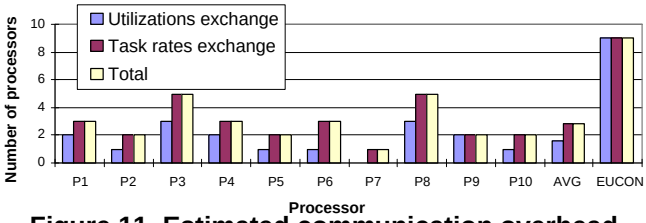


Figure 11. Estimated communication overhead

6.4 Scalability

Our final set of simulations evaluates the scalability of DEUCON in large systems. Figure 12 shows the direct neighborhood size, the number of concerned tasks, and the number of processors communicating with a controller under DEUCON as the number of processors increases from 100 to 1000 and the number of subtasks increases from 500 to 5000. Every result is the average value of all controllers in the system. Each task has 5 subtasks. All subtasks are randomly allocated to processors such that there are 5 subtasks on each processor. We can see that the size of direct neighborhood remains almost constant despite the ten-fold increase in the number of processors. At the same time, the number of concerned task and the number of processors communicating with a controller increase very slowly. Even in the system with 1000 processors, a controller only communicates with less than 34 processors. Therefore the per-controller overhead of DEUCON is almost independent of the total size of the system. This result indicates that DEUCON can scale effectively in large system. In addition, we observe that real-world systems may allocate subtasks in a clustered fashion, i.e., all subtasks of a subsystem tend to share several processors and only a small number of tasks run across multiple subsystems. We expect such clustered allocation to result in even smaller neighborhood size than the random allocation in our simulations.

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

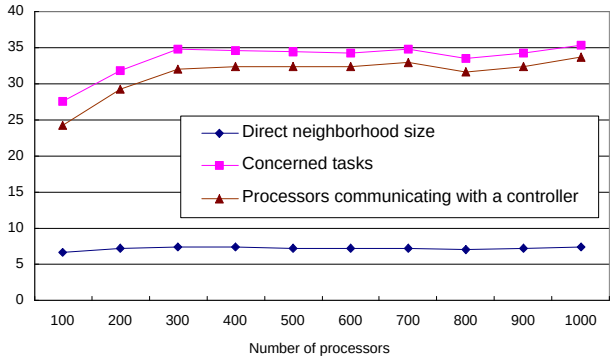


Figure 12. Scalability of DEUCON under Random Subtask Allocation

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 run-time. 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.

Acknowledgements

This research was supported in part by NSF CAREER award (grant CNS-0448554) and DARPA Adaptive and Reflective Middleware Systems (ARMS) program (grant NBCHC030140). We would also like to thank the reviewers for their detailed feedback.

References

[1] T. Abdelzaher, J. Stankovic, C. Lu, R. Zhang, and Y. Lu. Feedback performance control in sofware services. *IEEE Control Systems*, 23(3), June 2003.

[2] L. Abeni, L. Palopoli, G. Lipari, and J. Walpole. Analysis of a reservation-based feedback scheduler. In *IEEE RTSS*, Dec. 2002.

[3] S. Brandt, G. Nutt, T. Berk, and J. Mankovich. A dynamic quality of service middleware agent for mediating application resource usage. In *IEEE RTSS*, Dec. 1998.

[4] G. C. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni. Elastic scheduling for flexible workload management. *IEEE Trans. Comput.*, 51(3):289–302, 2002.

[5] M. Caccamo, G. Buttazzo, and L. Sha. Handling execution overruns in hard real-time control systems. *IEEE Trans. Comput.*, 51(7):835–849, 2002.

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

[7] E. Camponogara, D. Jia, B. Krogh, and S. Talukdar. Distributed model predictive control. *Control Systems Magazine*, 22(1):44–52, Feb. 2002.

[8] R. Carlson. Sandia SCADA program high-security SCADA LDRD final report. *SANDIA Report SAND2002-0729*, 2002.

[9] A. Cervin, J. Eker, B. Bernhardsson, and K.-E. Arzen. Feedback-feedforward scheduling of control tasks. *Real-Time Systems*, 23(1):25–53, July 2002.

[10] A. Goel, J. Walpole, and M. Shor. Real-rate scheduling. In *IEEE RTAS*, 2004.

[11] D. Henriksson and T. Olsson. Maximizing the use of computational resources in multi-camera feedback control. In *IEEE RTAS*, May 2004.

[12] B. Kao and H. Garcia-Molina. Deadline assignment in a distributed soft real-time system. *IEEE Trans. Parallel Distrib. Syst.*, 8(12):1268–1274, 1997.

[13] J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadline. In *IEEE RTSS*, 1990.

[14] F. Lewis and V. Syrmos. *Optimal Control, Second Edition*. John Wiley & Sons, Inc., 1995.

[15] S. Lin and G. Manimaran. Double-loop feedback-based scheduling approach for distributed real-time systems. In *HiPC*, pages 268–278, 2003.

[16] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of ACM, Vol. 20, No.1, pp. 46-61*, Jan. 1973.

[17] J. W. S. Liu. *Real-Time Systems*. Prentice Hall, 2000.

[18] C. Lu, J. Stankovic, G. Tao, and S. Son. Feedback control real-time scheduling: Framework, modeling, and algorithms. *Real-Time Systems*, 23(1/2):85–126, July 2002.

[19] C. Lu, X. Wang, and C. Gill. Feedback control real-time scheduling in ORB middleware. In *IEEE RTAS*, May 2003.

[20] C. Lu, X. Wang, and X. Koutsoukos. Feedback utilization control in distributed real-time systems with end-to-end tasks. *IEEE Trans. Parallel Distrib. Syst.*, 16(6):550–561, June 2005.

[21] P. Marti, G. Fohler, P. Fuertes, and K. Ramamritham. Improving quality-of-control using flexible timing constraints: metric and scheduling. In *IEEE RTSS*, 2002.

[22] M. D. Natale and J. Stankovic. Dynamic end-to-end guarantees in distributed real-time systems. In *IEEE RTSS*, 1994.

[23] R. Rajkumar, L. Sha, and J. P. Lehoczky. Real-time synchronization protocols for multiprocessors. In *IEEE RTAS*, Dec. 1988.

[24] D. Seto, J. P. Lehoczky, L. Sha, and K. G. Shin. On task schedulability in real-time control system. In *IEEE RTSS*, Dec. 1996.

[25] V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, and Z. Lu. Power-aware QoS management in web servers. In *IEEE RTSS*, 2003.

[26] C. Shen, K. Ramamritham, and J. A. Stankovic. Resource reclaiming in multiprocessor real-time systems. *IEEE Trans. Parallel Distrib. Syst.*, 4(4):382–397, 1993.

[27] J. A. Stankovic, T. He, T. Abdelzaher, M. Marley, G. Tao, S. Son, and C. Lu. Feedback control scheduling in distributed real-time systems. In *IEEE RTSS*, 2001.

[28] D. C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole. A feedback-driven proportion allocator for real-rate scheduling. In *Operating Systems Design and Implementation*, pages 145–158, 1999.

[29] J. Sun and J. W.-S. Liu. Synchronization protocols in distributed real-time systems. In *ICDCS*, 1996.

[30] X. Wang, C. Lu, and X. Koutsoukos. Enhancing the robustness of distributed real-time middleware via end-to-end utilization control. In *IEEE RTSS*, 2005.

[31] Y. Ye. *Interior Point Algorithms: Theory and Analysis*. John Wiley & Sons, Inc., 1997.

[32] Y. Zhu and F. Mueller. Feedback EDF scheduling exploiting dynamic voltage scaling. In *IEEE RTAS*, 2004.