Towards Controllable Distributed Real-Time Systems with Feasible Utilization Control

Xiaorui Wang, *Member*, *IEEE*, Yingming Chen, Chenyang Lu, *Member*, *IEEE*, and Xenofon D. Koutsoukos, *Senior Member*, *IEEE*

Abstract—Feedback control techniques have recently been applied to a variety of real-time systems. However, a fundamental issue that was left out is guaranteeing system controllability and the feasibility of applying feedback control to such systems. No control algorithms can effectively control a system which itself is uncontrollable or infeasible. In this paper, we use the multiprocessor utilization control problem as a representative example to study the controllability and feasibility of distributed real-time systems. We prove that controllability and feasibility of a system depend crucially on end-to-end task allocations. We then present algorithms for deploying end-to-end tasks to ensure that the system is controllable and utilization control is feasible for the system. Furthermore, we develop runtime algorithms to maintain controllability and feasibility by reallocating tasks dynamically in response to workload variations, such as task terminations and migrations caused by processor failures. We implement our algorithms in a robust real-time middleware system and report empirical results on an experimental test-bed. We also evaluate the performance of our approach in large systems using numerical experiments. Our results demonstrate that the proposed task allocation algorithms improve the robustness of feedback control in distributed real-time systems.

Index Terms—Real-time and embedded systems, distributed systems, feedback control, utilization control, controllability, feasibility.

1 Introduction

Recent years have seen increasing attention being given to applying feedback control techniques to real-time computing and communication systems (e.g., [1], [3], [10], [27], [8], [34]). In contrast to traditional approaches that rely on accurate knowledge about system workload, control-based solutions can provide robust Quality of Service (QoS) control in *unpredictable* environments by adapting to workload variations based on dynamic feedback. A particularly suitable application class for feedback control is *Distributed Real-time Embedded (DRE)* systems whose workloads are unknown and vary significantly at runtime. For example, task execution times in vision-based feedback control systems depend on the content of live camera images of changing environments [16]. Likewise, the supervisory control and data acquisition (SCADA) systems for power grid control may experience 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 bursty user requests and even cyber attacks [9], [8].

While existing feedback control work on real-time systems has shown promise in providing robust QoS guarantees in unexpected environments, several essential issues regarding feedback control have received little to no attention. A fundamental problem is guaranteeing system *controllability*. Controllability is an important property of DRE systems. No control algorithm can achieve its control objective if the system itself is uncontrollable. From the system perspective, uncontrollability is commonly caused by the lack of enough actuators in the system to provide complete control for all desired performance metrics. Along with controllability, it is also important to investigate the *feasibility* problem, which is caused by actuation constraints (e.g., rate constraints of periodic tasks in a DRE system). A controllable system may still fail to achieve its desired performance set points when its actuators saturate due to constraints. Therefore, both controllability and feasibility are important system properties and have to be guaranteed for DRE systems.

In this paper, we use utilization control as an example to study the controllability and feasibility of DRE systems. End-to-end utilization control [28], [37] has been demonstrated to be an effective way to guarantee the *end-to-end deadlines* of all periodic tasks in a soft DRE system, despite uncertainties in task execution times and coupling among processors. In end-to-end scheduling [25], the deadline of an end-to-end task is divided into subdeadlines of its subtasks, and so the problem of meeting the end-to-end deadline is transformed to the problem of meeting the subdeadline of each subtask. A well-known approach to meeting all subdeadlines on a processor is guaranteeing that the real CPU utilization of the processor remains below

- *X. Wang is with the Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, TN 37996-2100. E-mail: xwang@eecs.utk.edu.*
- *Y. Chen is with Microsoft Corporation, One Microsoft Way, Redmond, WA 98052-6399. E-mail: chen.yingming@gmail.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 11 Dec. 2007; revised 3 Sept. 2008; accepted 9 Dec. 2008; published online 14 Jan. 2009.
Recommended for acceptance by A. Zomaya.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2007-12-0635.
Digital Object Identifier no. 10.1109/TC.2009.19.

an appropriate *schedulable utilization bound*, under certain scheduling algorithms (e.g., RMS) [25]. In a real DRE system where the invocation rates of some tasks may be adjustable within certain ranges, selected tasks can run slower to lower the CPU utilization to the desired level when the system is having an unexpected execution time increase. It is usually preferable for DRE systems to control the utilizations of all the processors in the system to stay slightly below their schedulable bounds, so that undesired deadline misses can be avoided while higher task rates can be achieved. We assume that a higher task rate may contribute a higher value to the application at the cost of a higher CPU utilization. The assumption is reasonable because in many real-time applications, a task running at a higher rate usually can achieve better system QoS. For example, in a real-time video surveillance system, a higher sampling rate of the video camera can lead to an increased number of image frames transmitted to the monitoring center in every second, and thus an improved chance of detecting any intruder. Likewise, in real-time multimedia systems, a higher task rate can contribute to a higher video resolution [6]. *Therefore, as a result of utilization control, the value of the system can be maximized without causing undesired deadline misses* [26]. Utilization control can also be deployed in middleware systems to support Quality of Service portability [27], or enhance system survivability by providing overload protection against workload fluctuation [38].

In utilization control, an uncontrollable DRE system is a system for which it is impossible to find a sequence of task rates that take the utilizations of all the processors to the desired set points specified by the application. Uncontrollability may lead to deadline misses when the actual processor utilizations are higher than their set points. An infeasible system is interpreted as a system which fails to achieve its set points because the invocation rates of its tasks saturate at the boundaries of the allowed rate ranges. As a result of uncontrollability or infeasibility, some processors may become overloaded while some other processors may be poorly utilized at the same time. This kind of workload unbalance is highly undesirable for real-time systems. First, if any processor is overloaded, the consequent *deadline misses* may cause serious problems. Second, in some DRE systems, if any processor is underutilized, the system value is unnecessarily reduced because an adjustment to the system may easily enable all processors to achieve their desired utilization bounds and have higher task rates. Therefore, it is highly desirable to increase the system value by driving the processor utilizations close to the utilization bounds. In contrast, a commonly used naive solution is to simply keep the utilizations of all processors under their bounds, with no regard to controllability and feasibility. This naive solution would result in undesired low task rates, and thus, poor system value. With controllability and feasibility guarantees, we can maximize the system value by running all tasks at the highest possible rates without causing any deadline misses [26].

In this paper, we show that system controllability and feasibility can be guaranteed by adjusting certain system configurations, such as end-to-end task allocation. Specifically, the contributions of this paper are fivefold:

- We formulate the controllability and feasibility problem as an end-to-end task allocation problem.
- We design task allocation algorithms to ensure a system is controllable and feasible.
- We analyze the impact of workload variations on controllability and feasibility, and design efficient online algorithms to dynamically adjust task allocation.
- We integrate our algorithms with a robust real-time middleware to maintain system controllability and feasibility for deployed DRE applications.
- We present both empirical and numerical results to demonstrate the effectiveness of our algorithms.

The rest of this paper is structured as follows: We first review related work in Section 2. We then formulate the controllability and feasibility problems in Section 3. Section 4 analyzes the controllability problem to provide a theoretical foundation for algorithm design. Section 5 presents our offline task allocation algorithms. Section 6 introduces runtime analysis and online allocation adjustment algorithms. Section 7 discusses the implementation of the algorithms in a real-time middleware system. Section 8 presents our empirical and numerical results. Finally, Section 9 summarizes the paper.

2 Related Work

A survey of feedback performance control in computing systems is presented in [1]. Many projects that applied control theory to real-time scheduling and utilization control are closely related to this paper. Steere et al. and Goel et al. developed feedback-based schedulers [34], [14] that guarantee desired progress rates for real-time applications. Abeni et al. presented control analysis of a reservation-based feedback scheduler [3], [2]. Lu et al. developed a middleware service which adopts feedback control scheduling algorithms to control CPU utilization and deadline miss ratio [27]. Feedback control has also been successfully applied to power control [31], [21] and digital control applications [10]. For systems requiring discrete control adaptation strategies, hybrid control theory has been adopted to control state transitions among different system configurations [20].

Stankovic et al. [33] and Lin and Manimaran [23] proposed feedback control scheduling algorithms for *distributed* real-time systems with independent tasks. These algorithms do not address the interactions between processors caused by end-to-end tasks in DRE systems. Diao et al. developed a Multi-Input-Multi-Output (MIMO) control algorithm for load balancing in data servers [11]. However, their algorithm cannot handle actuation constraints which are also common in DRE systems. In contrast, our previous work EUCON [28] and DEUCON [37] are specially designed to handle the constrained MIMO utilization control problem for multiple processors that are coupled due to end-to-end tasks.

Both controllability and feasibility are important system properties wherever MIMO control is necessary. This paper presents the first study on the controllability and feasibility of DRE systems. Recently, Karamanolis et al. raised the problem of designing controllable systems [19]. However,

their paper focused only on some practical issues regarding how to get better control performance for Single-Input-Single-Output (SISO) systems. In contrast, our work investigates the fundamental issues defined in control theory, such as whether it is possible to control a DRE system and how to make an uncontrollable system controllable. Feasibility is another important issue. While the feasibility of scheduling tasks [5] has been addressed before in the real-time community, in this paper, we focus on the feasibility of controlling DRE systems.

We formulate the controllability and feasibility problem as a task allocation problem in DRE systems. Task allocation is a classical problem which has been discussed by many existing projects (e.g., [17], [13], [4]). The difference between our work and those related projects is that we are trying to guarantee system controllability and feasibility, instead of minimizing communication cost or ensuring load balancing.

3 Problem Formulations

In this section, we first introduce the system model employed in our work. We then formulate the controllability and feasibility problems.

3.1 System Model

We adopt an end-to-end task model [25] 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 set of subtasks $\{T_{ij}|1 \leq j \leq m_i\}$ which 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, i.e., subtask $T_{ij}(1 < j \leq m_i)$ 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 (e.g., release guard [35]) is used to enforce the precedence constraints, every subtask is released periodically without jitter. As a result, all the subtasks of a 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. 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 three important properties. First, each subtask T_{ij} has an *estimated* execution time c_{ij} available at design time. Note that c_{ij} is not necessarily the worst case execution time. The *actual* execution time of T_{ij} may be different from c_{ij} 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 (e.g., digital control [30], sensor update, and multimedia [6]) can be dynamically adjusted without causing system failure. Third, while each subtask is initially allocated (based on controllability and feasibility considerations) to a processor before a system runs, subtasks may be moved to other processors at runtime to maintain controllability and feasibility. The migration of subtasks is subject to resource availability. The precedence between subtasks has to be maintained during a migration.

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 [35], the deadline of an end-to-end task is divided into subdeadlines of its subtasks [18], [29]. 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 [22], [24]. For example, in our experiments, the subtasks on each processor are scheduled by the RMS algorithm. The subdeadline of each subtask T_{ij} can be set to be equal to the period of task T_i. As a result, the RMS utilization bound can be used on each processor to guarantee that all the subtasks can meet their respective subdeadlines. Note that our system model is not limited to a certain scheduling algorithm. Other utilization bounds can also be used by our system model when the subdeadline of each subtask is not equal to its period. Utilization control is not designed to handle network delays. Network delay may be handled by treating each network link as a processor [35], or by considering the impact of worst case network delay in subdeadline assignment.

In our previous work [28], we have modeled the utilization control problem by establishing difference equations to capture the dynamics of a DRE system with n processors and m end-to-end periodic tasks. The utilization controller 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 are the changes 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 DRE system is described by the following MIMO model:

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

The *subtask allocation matrix*, $\mathbf{F}$, is an $n \times m$ matrix, where $f_{ij} = \sum_{T_{jl} \in S_{ij}} c_{jl}$ 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. $\mathbf{G} = diag[g_1 \dots g_n]$, where g_i represents the ratio between the actual utilization change and its estimation. For example, $g_i = 1$ means that the actual utilization change of processor P_i is exactly equal to its estimated utilization change. The exact value of g_i is *unknown* due to the unpredictability in execution times. Note that $\mathbf{G}$ describes the effect of workload uncertainty in a DRE system. Our model is *not* limited to constant $\mathbf{G}$. When $\mathbf{G}$ varies along time, we can identify a range of $\mathbf{G}$ for which there exists a common Lyapunov function for all $\mathbf{G}$s. As a result, our model can handle time-varying $\mathbf{G}$ without any change. A detailed discussion about time-varying $\mathbf{G}$ is available in [37].

As an example, the DRE system shown in Fig. 1 is modeled by (1) with the following parameters:

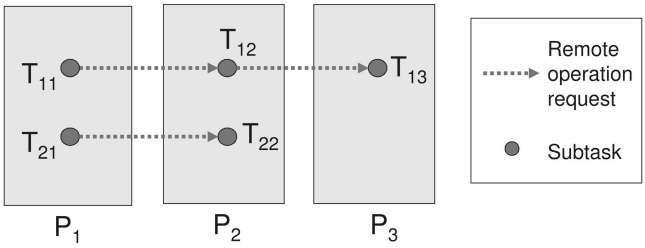

Fig. 1. An example DRE system.

$$\mathbf{u(k)} = \begin{bmatrix} u_1(k) \\ u_2(k) \\ u_3(k) \end{bmatrix}, \quad \mathbf{G} = \begin{bmatrix} g_1 & 0 & 0 \\ 0 & g_2 & 0 \\ 0 & 0 & g_3 \end{bmatrix},$$
$$\mathbf{F} = \begin{bmatrix} c_{11} & c_{21} \\ c_{12} & c_{22} \\ c_{13} & 0 \end{bmatrix}, \quad \mathbf{\Delta r(k)} = \begin{bmatrix} \Delta r_1(k) \\ \Delta r_2(k) \end{bmatrix}.$$

3.2 Controllability Problem

In an MIMO control system, if a sequence of control input variables can be found that take all control output variables from any initial conditions to any desired final conditions in a finite time interval, the MIMO system is said to be *controllable* [12]; otherwise, the system is uncontrollable. According to the control theory, an MIMO system $\mathbf{x(k+1)} = \mathbf{\Phi x(k)} + \mathbf{\Gamma v(k)}$ with n control outputs $[x_1(k) \dots x_n(k)]$ and m control inputs $[v_1(k) \dots v_m(k)]$ is controllable iff the rank of its *controllability matrix* $\mathbf{C} = [\mathbf{\Gamma\ \Phi\Gamma\ \dots\ \Phi^{n-1}\Gamma}]$ is n, the order of the system [12].

Definition. *A DRE system is* controllable *if there exists a sequence of task rates that take the utilizations of all processors in the system to any desired utilization set points.*

In our system model (1), we assume that matrix $\mathbf{G}$ is the identity matrix $diag[1 \dots 1]$ because system gains g_i are unknown at design time [28]. That means, we assume the estimated execution times are accurate at design time. Although this approximate model is not an exact characterization of the real system, we will show later that system gains do *not* affect system controllability. Hence, the controllability matrix of the system model is an $n \times nm$ matrix $\mathbf{C} = [\mathbf{F\ F\ \dots\ F}]$. In order to have a controllable DRE system, we have to guarantee the rank of the controllability matrix $\mathbf{C}$ is n, the number of processors in the system.

3.3 Feasibility Problem

In control theory, the condition of controllability is based on the assumption that there are no actuation constraints (i.e., rate constraints). However, as introduced in our task model, the rate of each task T_i can only be adjusted within a range $[R_{min,i}, R_{max,i}]$, namely $R_{min,i} \le r_i \le R_{max,i}, (1 \le i \le m)$. Therefore, a system proved to be controllable may still not be able to achieve the desired utilization set points, as the task rates may saturate.

Definition. *If a controllable DRE system cannot get to the set points because the rates of one or more of its tasks saturate at the rate boundaries, we say it is* infeasible *to achieve the set points for the system. Otherwise, we say utilization control is* feasible *for the system.*

In utilization control, although a feasible system is preferable, it is much more important to keep the processor utilizations below the desired set points. The reason is that overload may cause *deadline misses*, and thus, is much more undesirable than underutilization in DRE systems. Therefore, in this paper, we focus on *practical feasibility* defined below.

Definition. *Utilization control is* practically feasible *for a DRE system whose task rate constraints allow the utilizations of all processors to either get to or stay below the desired set points.*

An effective solution to the feasibility problem is subtask allocation adjustment. For instance, if a processor in the system remains overloaded because all its subtasks already reach their lower rate boundaries, we may move one subtask away from the processor (subject to resource availability on other processors) so that it can have less workload and then recover from overload. While this solution is sufficient for systems where execution times never change, it has to be extended for DRE systems whose execution times may vary unpredictably. In such systems, a previously feasible system may become infeasible at runtime. Continuously monitoring feasibility and migrating subtasks would introduce large runtime overhead. Hence, instead of guaranteeing a system to be feasible for certain execution times, we try to increase the likelihood of the system remaining feasible even under variations, so that we can reduce the overhead of moving subtasks later at runtime.

We first introduce several definitions:

Definition. *The* minimum estimated utilization *of processor P_i is defined as the sum of the products of the estimated execution times and the minimum allowed rates of all subtasks on the processor. Specifically, $u_{min,i} = \sum_{T_{jl} \in S_i} c_{jl} R_{min,j}$, where S_i represents the set of subtasks located on processor P_i.*

Definition. *The difference between B_i, the set point of processor P_i, and its minimum estimated utilization is defined as its* feasibility margin. *Specifically,* $\text{margin} = B_i - u_{min,i}$.

When the variations of execution times cause the utilization of P_i to deviate from its set point B_i, a large feasibility margin can give task rates enough space for rate adaptation so that the utilization can reconverge to the set point. Hence, we want to adjust subtask allocations so that the task rates can stay as far away from their lower boundaries as possible when processors settle at their set points. In other words, our goal is to maximize the feasibility margin for all the processors in order to maximize the chance of having a feasible system under variations. If we consider a DRE system infeasible when any processor becomes infeasible, the feasibility problem becomes a problem of maximizing the smallest feasibility margin among all the processors in the system. Hence, the feasibility problem can be formulated as finding a subtask allocation to optimize the following objective:

$$\max(\min_{1 \le i \le n}(\left|B_i - u_{min,i}\right|)). \tag{2}$$

This optimization problem is subject to two constraints. The first one is a utilization constraint. The minimum

estimated utilization $u_{min,i}$ of each processor P_i is not allowed to be larger than B_i, otherwise, the system would be infeasible based on the estimated execution times. The second one is a resource constraint. As a common practical issue in DRE systems, each subtask can *only* be allocated to a specific set of processors due to resource availability. Note that the set point B_i of each processor P_i is a function of its number of subtasks when the system is scheduled by some algorithms like RMS [24].

4 Controllability Analysis

In this section, we present the theoretical analysis of the controllability problem, which provides a foundation for the design of our task allocation algorithms.

4.1 Controllability Condition

We first analyze the controllability matrix to see how we can guarantee its rank to be equal to n, the number of processors in the system.

Theorem 4.1. *A DRE system is controllable if and only if the rank of its subtask allocation matrix* $\mathbf{F}$ *is* n.

Proof. We prove that the rank of the subtask allocation matrix $\mathbf{F}$ is equal to the rank of the controllability matrix $\mathbf{C} = [\mathbf{F}\ \mathbf{F}\ \ldots\ \mathbf{F}]$. We first transform $\mathbf{C}$ to a matrix $\mathbf{C}' = [\mathbf{F}\ 0\ \ldots\ 0]$ by subtracting every column of the first $\mathbf{F}$ from the rest $\mathbf{F}$s. Since elementary transformations do not change the rank of a matrix, $\mathbf{C}$ has the same rank as $\mathbf{C}'$. Clearly, $\mathbf{C}'$ has the same rank as $\mathbf{F}$. Hence, the system is controllable if and only if the rank of $\mathbf{F}$ is n. □

Example. The DRE system shown in Fig. 1 is not controllable because the rank of its subtask allocation matrix $\mathbf{F}$ is 2, while there are three processors in the system.

Corollary 4.2. *A DRE system with* n *processors and* m *end-to-end tasks is* uncontrollable *if* $m < n$.

In other words, any DRE system must have at least as many tasks (control inputs) as processors (control outputs) in order to be controllable for utilization control. Note that $m \geq n$ is a necessary but not sufficient condition of controllability. When this condition is met, a system is not necessarily controllable. However, as we will show later, we may adjust the subtask allocation matrix of the system to make it controllable. Hence, through task allocation, a system can achieve both feasibility and controllability. Note that when there are not enough tasks (i.e., $m < n$), we can easily use *fewer* processors to run the same DRE applications so that the system becomes controllable and the system value could be maximized [26] with less computing resource. As discussed before, *a controllable DRE system can have higher task rates, and thus, higher system value without deadline misses*. The naive solution of simply keeping the utilizations of all processors under their bounds, with no regard to controllability, would result in undesired low task rates, and thus, poor system value.

A potential price we may have to pay is that the increased CPU utilization may cause the system to be less robust against large instantaneous execution time variations. There exists an inevitable trade-off between system robustness and system efficiency. With effective controllability maintenance and increased feasibility margins, our combined controllability and feasibility strategy can achieve desired CPU utilizations for improved system efficiency while maximizing the system robustness against execution time variations for soft DRE systems.

4.2 Structural Controllability

As the algorithms proposed in this paper are used in DRE systems, here we narrow down our attention from *complete controllability* (i.e., controllability defined before) to *structural controllability* [32]. A system is structurally controllable if there exists another system which is structurally equivalent to the system and is completely controllable [32]. Two systems are structurally equivalent if there is a one-to-one correspondence between the locations of the fixed zeros and nonzero items in their controllability matrices [32].

A structurally controllable system may not always be controllable because the elements of two rows/columns of its controllability matrix could happen to be proportional, which causes its rank to be smaller than the system order. In our system model, two rows are proportional meaning thereby that the subtasks on two processors belong to exactly the same set of tasks and the execution times of corresponding subtasks are strictly proportional to each other. Two columns are proportional means that two tasks are deployed on exactly the same set of processors and the execution times of their subtasks on each processor are strictly proportional to each other. In general, finding proportional rows and columns is computationally expensive [32]. Fortunately, in DRE systems, such cases are very rare due to the high variations in task execution times on modern processors. Therefore, in practice, we can easily identify potentially proportional rows and columns in the allocation matrix based on the configuration of DRE applications. Once we identify a set of proportional rows or columns, we combine them as a single row or column for our analysis. As a result, structural controllability ensures controllability in the modified allocation matrix. As discussed later, our analysis and algorithms can ensure structural controllability (and thus, controllability). Hence, we use controllability and structural controllability interchangeably hereinafter.

5 Offline Task Allocation Algorithms

The solutions to both the controllability and feasibility problems rely on subtask allocation. In this section, we propose a two-step approach to allocate subtasks in a DRE system. The first algorithm aims to increase the feasibility margin. The second algorithm ensures controllability by adjusting the allocation while minimizing the influence on the feasibility margin.

5.1 Increasing Feasibility Margin

As suggested by the optimization objective in (2), the feasibility problem is related to both load balancing [4] and variable-size bin packing [25]. It is related to the variable-size bin packing problem because it needs to pack all subtasks to processors and the capacity of a processor shrinks when its number of subtasks increases. It differs

```
(1) Enqueue all subtasks T_jl in the order of decreasing u_min,jl;
(2) While there is at least one subtask in the queue,
      Pop up the first subtask T_jl (which has the largest u_min,jl);
      For each processor P_q = cons[T_jl, q++],
        If u_current,q + u_min,jl ≤ B_q,
          u_new,q = u_current,q + u_min,jl;
          Feasibility margin of P_q = B_q − u_new,q;
        Endif;
      Endfor;
      Allocate T_jl to processor P_i with the largest feasibility margin;
      If T_jl cannot be allocated to any processor,
        Algorithm fails;
    Endwhile.
```

Fig. 2. Pseudocode of the algorithm to increase feasibility margin.

from bin packing because the goal here is to maximize the feasibility margin by distributing subtasks to different processors, instead of using fewest processors. The problem is closer to the load balancing problem but the difference is that we are trying to maximize the smallest feasibility margin instead of minimizing the highest utilization among all processors. Clearly, our problem can be reduced to the load balancing problem, which is an NP-hard problem [7]. Here, we present a heuristic feasibility algorithm to iteratively allocate the subtask with the current largest minimum estimated utilizations to processors. We note that our algorithm shares the same basic idea with an existing Max-Min algorithm used for load balancing [4]. However, our algorithm is designed to maximize the smallest feasibility margin, which is a totally different problem from load balancing. Furthermore, our algorithm can handle resource and utilization constraints as discussed later.

In our feasibility algorithm, we first sort all subtasks based on their minimum estimated utilization, $u_{min,jl} = c_{ij}R_{min,j}$. We then pick the subtask with the currently largest $u_{min,jl}$ and allocate it to the processor that has the largest feasibility margin after this allocation. We continue the process until all the subtasks are allocated. Note that the allocation at each step is subject to both the utilization and resource constraints. A subtask can *only* be allocated to a processor where the required resource is available. The utilization constraint is checked at each step when a subtask is allocated to a processor. If the largest feasibility margin after allocating a subtask to the system becomes negative, the algorithm fails. In that case, more advanced algorithms, such as Mixed Integer Programming, may be adopted to provide a solution at a cost that could be comparable to the cost of exhaustive search [4].

The detailed algorithm is shown in Fig. 2. The resource constraints are represented by an $s \times p$ matrix $cons$, where s is the total number of subtasks in the system and p is the number of processors on which a subtask can execute. Each element $cons[T_{jl}, q]$ is the qth processor that the subtask T_{jl} can be allocated to. We assume that all processors are homogeneous here, but the algorithm can be easily extended to systems with heterogeneous processors.

Now we analyze the complexity of this algorithm. The complexity of step 1 is $O(s \log s)$, where s is the total number of subtasks in the system. The complexity of step 2 is sp, where p is the number of processors that a subtask can be allocated to. Hence, the time complexity of the feasibility algorithm is $O(\max(s \log(s), sp))$.

5.2 Ensuring Controllability

After our feasibility algorithm successfully allocates all subtasks, we check the allocation matrix $\mathbf{F}$ to determine whether the current workload configuration is controllable. If it is, the workload is accepted for deployment on the target DRE system. Otherwise, we process the workload with a novel controllability adjustment algorithm. In the algorithm, for every processor, we search all tasks that have subtasks on the processor to find one task to *dedicate* to the processor. The task is called the *dedicated task* of the processor and its subtasks on the processor are called the *dedicated subtasks*. A task can only be dedicated to one processor. For those processors which fail to find dedicated tasks, we migrate subtasks of some nondedicated tasks from other processors to them so that they can have those tasks dedicated to them.

Theorem 5.1. *If every processor in a system has a dedicated task, the system is controllable.*

Proof. If every processor has a dedicated task, the allocation matrix can be proved to have full rank (i.e., its rank equals the order of the system). To prove that, we can move the columns of the matrix so that all tasks can place their dedicated subtasks on the diagonal of the allocation matrix. As described in Section 3.5 for structural controllability, there are no two rows or columns that are proportional to each other in the matrix. As a result, a matrix has full rank if there is no zero on its diagonal. Hence, a system is guaranteed to have controllability if every processor has a dedicated task. □

Note that Theorem 5.1 is both a sufficient and a necessary condition for controllability. The rationale behind dedicating tasks to processors can also be explained from the system perspective, i.e., that each processor can rely on the rate adaptation of its dedicated task to achieve its utilization set point assuming there are no rate constraints.

Our controllability algorithm uses two auxiliary matrices $\mathbf{E}$ and $\mathbf{B}$. The algorithm first initializes the two matrices and

(1) Create two $n \times m$ auxiliary matrices $\mathbf{E}$ and $\mathbf{B}$;
$\mathbf{E}$ is used to calculate $u_{min,jl} = c_{jl} * R_{min,j}$ for every existing subtask T_{jl} on each processor;
$\mathbf{B}$ is used to find the best candidate subtask for each processor;
Initialize all elements in the auxiliary matrix $\mathbf{E}$ to be zero;
For every non−zero element $\mathbf{F}[i,j] = c_{jl}$ in the allocation matrix $\mathbf{F}$, $\mathbf{E}[i,j] = u_{min,jl}$;
Initialize all elements in the auxiliary matrix $\mathbf{B}$ to be the maximum integer;
Sort all the processors in the increasing order of number of subtasks;

(2) For every column (task) in $\mathbf{E}$,
Find the subtask T_{jl} such that $u_{min,jl} = min(u_{min,jl'}\ (0 \le l' \le m_j))$;
T_{jl} is the subtask with the smallest minimum estimated utilization in task T_j;
T_{jl} is hence the best candidate if we need to move a subtask of task T_j to a processor;
For each processor P_q that is allowed to execute T_{jl} based on the constraints matrix $cons$,
if $\mathbf{F}[q,j] = 0$, $\mathbf{B}[q,j] = u_{min,jl}$;
Endfor;
Endfor;

(3) For each row of $\mathbf{E}$, sort this row in the decreasing order of $u_{min,jl}$;
For each row of $\mathbf{B}$, sort this row in the increasing order of $u_{min,jl}$;

(4) For each row (processor P_i in the increasing order of number of subtasks) in $\mathbf{E}$,
For each subtask T_{jl} that is sorted in Step 3,
If task T_j is not dedicated, dedicate T_j to P_i and exit the inner loop;
Endfor;
If all the current subtasks in the row of P_i are already dedicated to other processors,
In the corresponding row (processor P_i) in $\mathbf{B}$,
For each subtask T_{jl} that is sorted in Step 3,
If task T_j is not dedicated,
Move the best candidate subtask T_{jl} to P_i;
Dedicate task T_j to P_i;
Adjust the allocation matrix $\mathbf{F}$ accordingly and exit the inner loop;
Endif;
Endfor;
If cannot find a non−dedicated task, algorithm fails;
Endif;
Endfor. // for each row

Fig. 3. Pseudocode of the algorithm to ensure controllability.

sorts all the processors based on their numbers of subtasks. The algorithm will later dedicate tasks to the processors with fewer subtasks first because that may reduce the necessity of moving subtasks. The second step preprocesses the auxiliary matrices to speed up the later dedicating step. For every processor/task pair in the allocation matrix, we search for a candidate subtask by assuming that the processor fails to find its dedicated task and needs a subtask of this task to be moved to the processor. Since subtask migration may affect the feasibility margin of a system, we want to minimize the impact by moving the *best candidate subtask*, which has the smallest minimum estimated utilization and is allowed by the resource constraints to run on the target processor. Hence, the elements in the auxiliary matrix $\mathbf{B}$ are set to be the best candidate subtasks. The information will speed up the search process if a processor loses its dedicated task and needs to find a new one at runtime. In the third step, we sort all the existing subtasks of each processor in the auxiliary matrix $\mathbf{E}$ based on their minimum estimated utilizations. In the auxiliary matrix $\mathbf{B}$, we sort the best candidate subtasks of each processor based on their minimum estimated utilizations. The reason for sorting them is also to speed up the search process, which is especially important for extending the algorithm to support online task reallocation (as described in Section 6). In the last step, we start the dedicating process. If no task can be dedicated to a processor, we move the best candidate subtask of the first nondedicated task to the processor. This subtask is guaranteed to have the smallest minimum estimated utilization and so should only cause small impact on the system feasibility margin. The detailed algorithm is shown in Fig. 3.

Now we analyze the time complexity of this algorithm. The complexity of the four steps are $O(\max(nm, n \log n))$,

$O(nm)$, $O(nm \log m)$, and $O(nm)$, respectively. Hence, the time complexity of the whole controllability algorithm is $O(\max(n \log n, nm \log m))$.

6 Runtime Analysis and Adjustments

As every DRE system designed with feedback control should be configured to be controllable, and feasible offline before deployment, controllability and feasibility are mostly offline configurations. However, the subtask allocation matrix of a system may occasionally change at runtime due to workload variations, such as task termination. As a result, a workload processed with the offline algorithms may become uncontrollable or infeasible. Hence, controllability and feasibility may need to be maintained at runtime. While the algorithms presented in the previous section could be directly used to ensure controllability and feasibility effectively at runtime, they adjust the whole workload, and hence, may introduce higher computation overhead than can be tolerated at runtime. Although the overhead is acceptable for offline configurations, more efficient algorithms need to be developed to incrementally adjust only a small portion of the workload at runtime. In this section, we first analyze the impact of typical workload variations, and then, present online allocation adjustment algorithms to maintain controllability and feasibility to minimize the runtime cost.

6.1 Impact of Workload Variations

In DRE systems, workload variations may happen at runtime and change subtask allocations, which, in many ways, affect system feasibility or controllability. Hence, it is necessary to investigate their possible impact on system feasibility and controllability. In this paper, we focus on four common types of workload variations: task arrival, task termination, processor failure, and execution time variation. We analyze the possible impact of each type of variation on controllability as well as on feasibility. If a type of variation does not affect controllability or feasibility, we call it *harmless* to controllability or feasibility. Otherwise, we say it is *harmful*. The categorization of harmless and harmful variations allows us to execute our runtime adjustment algorithms *only* when harmful variations happen, so we can minimize the runtime overhead.

We first analyze the impact of workload variations on controllability.

Theorem 6.1. *Task arrival in a DRE system is harmless to controllability.*

Proof. is equivalent to adding a new column to the subtask allocation matrix $\mathbf{F}$, which does not reduce the rank of $\mathbf{F}$. □

Therefore, if the system is controllable, it remains controllable after task arrivals.

Theorem 6.2. *Task termination in a DRE system is harmful to controllability.*

Proof. Removing a column from the allocation matrix may reduce the rank of the matrix. □

TABLE 1
Impact of Different Types of Workload Variations

Variations	*Feasibility*	*Controllability*
Task arrival	harmful	harmless
Task termination	harmless	harmful
Processor failure	harmless	conditionally harmful
Exec time variation	harmful	harmless

Theorem 6.3. *Processor failure is harmful to controllability if the failed processor has more than $m - n + 2$ subtasks, where m and n are the numbers of tasks and processors, respectively.*

Proof. Removing a failed processor from a DRE system leads to removing a row from the allocation matrix $\mathbf{F}$. In a system with no fault tolerance mechanisms, all tasks having subtasks on the failed processor may terminate. Therefore, the processor failure may also result in removing several columns from the allocation matrix. If the rank of matrix $\mathbf{F}$ is originally n, any of its submatrices with size as $n' \times m'$ has a rank of $\min(n', m')$. We assume that after the processor failure, the allocation matrix has a rank of $\min(n-1, m'')$. In order for the matrix to have a rank less than $n-1$, we need to have $m'' \leq n-2$. Hence, we need to terminate at least $m - m'' = m - n + 2$ tasks. □

Note that in a system with certain fault tolerance mechanisms, the subtasks on the failed processor may be dynamically moved to other processors in the system. As a result, the processor failure will only lead to removing a row from the allocation matrix $\mathbf{F}$, which does not reduce the rank of the matrix. Therefore, processor failure is only conditionally harmful to system controllability.

Execution time variation is harmless to controllability because it does not change the structure of the controllability matrix. The impact of different types of workload variations on controllability is summarized in Table 1.

We now investigate feasibility by finding which types of variation may reduce the feasibility margin of a system. Clearly, any variation that increases system workload may cause the feasibility margin to decrease. Therefore, execution time variation and task arrival are harmful to feasibility because they may increase the workload of some processors in the system. Task termination reduces the workload of some processors, so it is harmless. Processor failure may cause task termination, so it is also a harmless variation to feasibility. The impact of different types of workload variations on feasibility is also summarized in Table 1.

6.2 Feasibility Adjustment

According to Table 1, two types of variations may reduce the feasibility margin of a system. Among them, execution time variation has been handled by the feasibility margin, which is designed to tolerate possible variations to the maximum degree, so that we can avoid the overhead of precisely measuring task execution times to check feasibility at runtime. To minimize the impact of task arrivals on feasibility and reduce runtime cost at the same time, we run our feasibility algorithm incrementally only to allocate new

```
(1) Remove the terminated task from the allocation matrix;
(2) If this task is not dedicated to a processor,
      Algorithm successfully ends;
(3) Else,
      For the processor that the terminated task was dedicated to,
        Run step 4 (Fig. 3) to find a dedicated task;
    Endif.
```

Fig. 4. Pseudocode of the algorithm to maintain controllability online.

tasks for a balance between the two conflicting goals. The algorithm presented in Fig. 2 is adopted to sort and allocate only the new arriving tasks. Hence, the computation overhead is now only $O(\max(qn\log(qn), qnp))$, where q is the number of arriving tasks. We acknowledge that the online adjustment algorithm is relatively simple, but we argue that short computation time is more important than the solution quality at runtime. More sophisticated algorithms (e.g., run the offline feasibility algorithm to reallocate all subtasks for the whole system) may significantly increase the runtime overhead with just a small gain of solution quality. Note that the online adjustment algorithm can be combined with a more sophisticated algorithm for a middle ground solution. While the online adjustment algorithm can be used to handle the workload changes immediately, the more sophisticated algorithm can run on a spare processor (if available) to find a better solution, which can be applied to the system at a later time.

6.3 Controllability Maintenance

According to Table 1, there are two situations that may jeopardize the system controllability: task termination and processor failure. The reason that processor failure is harmful is that it may cause one or more tasks to terminate. Hence, we only need to check and maintain controllability when tasks terminate, which can be handled incrementally by the runtime subtask reallocation algorithm shown in Fig. 4. Note that runtime subtask reallocation is still subject to resource availability as discussed before. The precedence constraint between different subtasks in a task is maintained during migration by having idle backup subtasks running on processors with available resource. We acknowledge that runtime task migration may have nonnegligible costs. However, we argue that system uncontrollability may lead to a much higher cost (e.g., deadline misses) than task migration, and thus, has to be handled despite possible costs. Detailed discussion regarding task migration implementation is available in Section 7. The time complexity of the controllability maintenance algorithm is $O(m)$, where m is the number of tasks in the system.

7 Middleware Implementation

Both the controllability and feasibility algorithms have been integrated in the FC-ORB middleware [38]. FC-ORB implements an end-to-end utilization control algorithm called EUCON [28]. Like any other feedback utilization control algorithm developed for DRE systems, the EUCON algorithm may experience controllability and feasibility problems and is used as an example platform to demonstrate the effectiveness of our algorithms. The two algorithms are integrated with the FC-ORB controller, which is running on a different processor from the controlled system.

Fig. 5 illustrates the implementation of the example DRE application shown in Fig. 1 in the middleware architecture of the extended FC-ORB system. Each subtask is executed by a separate thread whose priority is decided by a priority manager based on the real-time priority of the subtask. As a result, preemptive scheduling is enforced by the underlying operating system. As shown in Fig. 5, the first subtask of a task is implemented with a periodic timer. The timer periodically triggers a local operation (e.g., a method of an object) which implements the functionality of this subtask. Following the execution of this operation, a one-way remote operation request is pushed to the succeeding subtask that is located on another processor through the remote request lane. Each pair of preceding and succeeding subtasks maintains a separate TCP connection to avoid priority inversion in the communication subsystem. The release guard protocol is implemented using one-shot timers to enforce that the interval between two successive invocations of a same subtask is lower bounded by its period. Earlier research has shown that the release guard protocol can effectively reduce the end-to-end response time and jitter of tasks in DRE systems [35]. An end-to-end real-time task is finished when the execution of its last subtask is finished.

The controllability maintenance algorithm is implemented as a controllability handler. Based on our analysis in the previous section, task termination affects the controllability of a system. Consequently, the controllability handler is invoked whenever a task terminates at runtime. When that happens, the handler removes the terminated tasks from the control model, and then moves proper subtasks to maintain system controllability. After that, the handler reinitializes the controller and resumes the feedback control loop. Similarly, the feasibility adjustment algorithm has been implemented as a feasibility handler to do incremental subtask allocation whenever new tasks are admitted to the system.

To support online task reallocation, we extended FC-ORB to handle subtask migrations demanded by the controller. Similar to the *COLD PASSIVE* replication style used in Fault-Tolerant CORBA (FT-CORBA) [15], all subtasks are assumed to be stateless (except the connections between subsequent subtasks which are maintained by the middleware) so that the overhead of active state synchronization is avoided. The migration mechanism works as follows: Each subtask can have a primary instance and a few backup instances on the processors where it has the required resource. In the normal

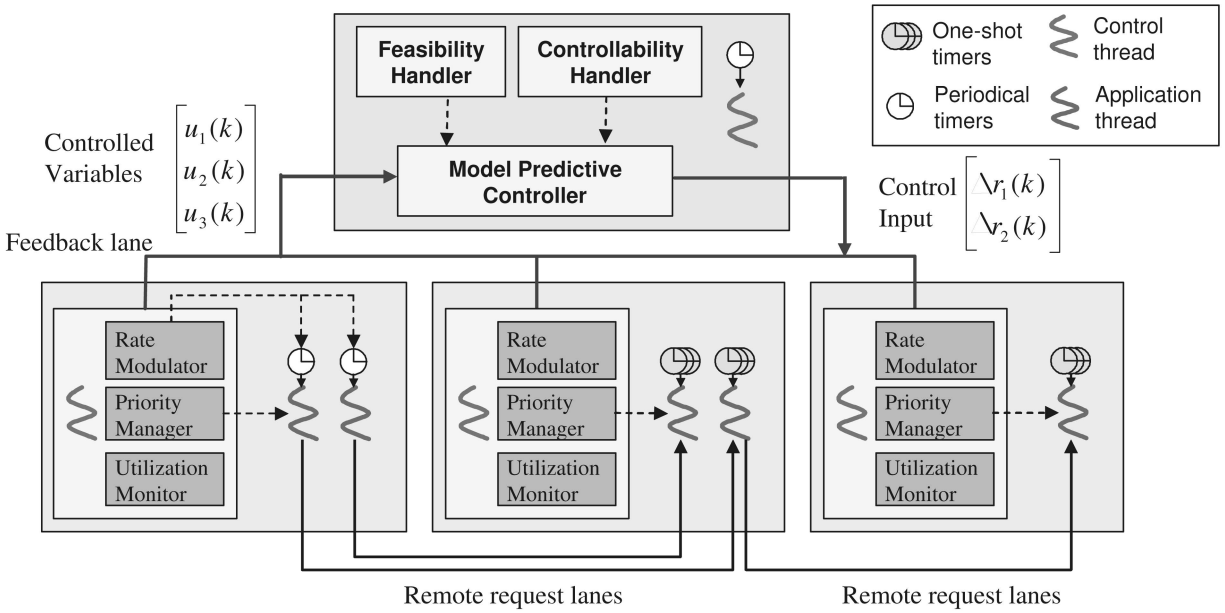

Fig. 5. Middleware architecture of the extended FC-ORB system.

mode, each subtask pushes remote operation requests only to the primary instance of its successor. As a result, the backup instances do not receive any requests and their threads remain idle. After a task migration decision is made by the controller, the predecessor of the migrated subtask switches the connection to the desired backup instance and sends the remote operation requests to it. In the case when the first subtask of a task has to be moved, the controller activates the proper backup instance of the subtask.

The backup subtasks are also used to enhance the system robustness by handling persistent single-processor failures. The failover mechanism works as follows: After a processor fails, the predecessor of a subtask located on the failed processor detects the communication failure based on the underlying socket read/write errors. The predecessor immediately switches the connection to the backup instance of its successor and sends the remote operation requests to it. In the case when the failed processor hosts the first subtask of a task, the controller activates the backup instance of the subtask. Consequently, the execution of the end-to-end tasks is resumed after a transient interruption.

8 Experiments

In this section, we present the results of two sets of experiments. First, we present empirical results based on the extended FC-ORB middleware system to demonstrate the effectiveness of the online algorithms. Second, we evaluate the offline subtask allocation algorithms using numerical experiments, which allow us to use a large number of randomly generated workloads to stress test the algorithms in large systems.

8.1 Empirical Results

In this section, we present the experiments conducted on a real DRE system that are implemented, based on the extended FC-ORB middleware. We first introduce the experimental configurations. We then present the experimental results on controllability and feasibility, respectively, by contrasting systems with and without the dynamic algorithms.

8.1.1 Experimental Setup

We perform our experiments on a test bed of six PCs. All applications and the ORB service run on four Pentium-IV PCs (P_1-P_4) and one Celeron PC (P_5). P_1 and P_4 are 2.80 GHz while P_2 and P_3 are 2.53 GHz. P_1-P_4 all are equipped with 512 KB cache and 512 MB RAM. P_5 is 1.80 GHz and has 128 KB cache and 512 MB RAM. All application PCs run RedHat Linux 2.4.22. The controller is located on another Pentium-IV 2 GHz PC with 512 KB cache and 256 MB RAM. The controller PC runs Windows XP Professional with MATLAB 6.0. P_1-P_4 are connected via an internal switch and communicate with P_5 and the controller PC through the departmental 100 Mbps LAN.

Our experiments run a medium-sized workload that initially comprises seven end-to-end tasks (T_1-T_7), with a total of 18 subtasks. Fig. 6a shows how the seven tasks are distributed on the five application processors. The detailed workload parameters are shown in Table 2. For example, task T_4 has two subtasks $T_{4,1}$ and $T_{4,2}$ running on processors P_2 and P_1 with an estimated execution time of 25 and 24 ms, respectively. Three new tasks (T_8-T_{10}) are admitted in an experiment to the system to test system feasibility. The subtasks on each processor are scheduled by the RMS algorithm [24]. Each task's end-to-end deadline is $d_i = m_i/r_i(k)$, where m_i is the number of subtasks in task T_i, and $r_i(k)$ is the current rate of 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 utilization set point of every processor is set as 0.7 because 1) 0.7 is the RMS schedulable utilization bound with 25 subtasks [24], and hence, can ensure that the processors have no

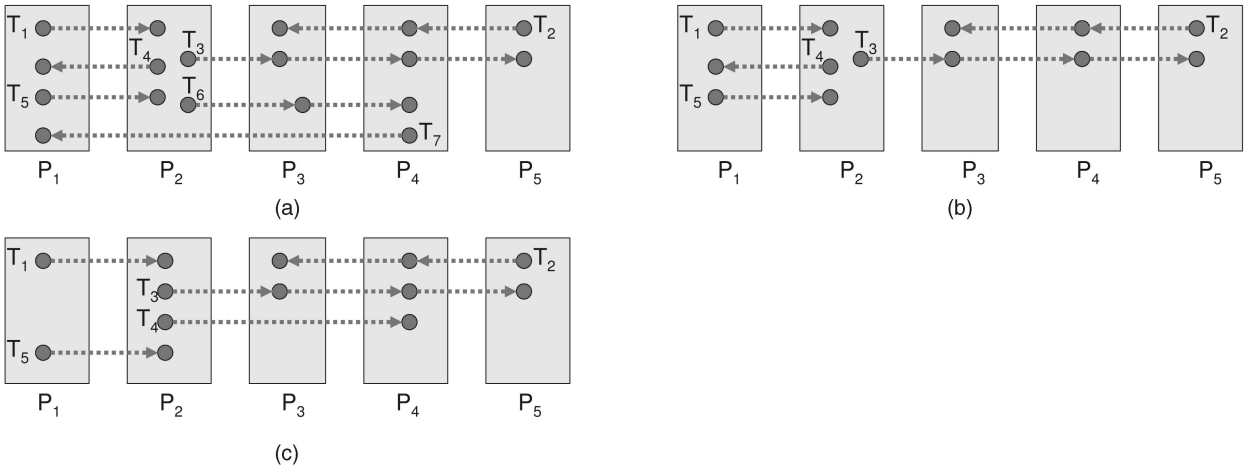

Fig. 6. Workload configuration and variations in controllability experiments. (a) Initial task allocation. (b) Allocation after task termination. (c) Allocation after controllability maintenance.

deadline miss due to fact that they always have fewer than 25 subtasks; and 2) we let all processors have the same set point so that it is easier to observe whether their utilizations can converge to the set point when we present the results of all processors in the same figure later. All (sub)tasks meet their (sub)deadlines if the desired utilization on every processor is enforced. The sampling period of the utilization control service is $T_s = 5$ seconds.

8.1.2 Controllability

In our first experiment, we run the original FC-ORB with an initial workload shown in Fig. 6a. The rates of all tasks in the workload are selected based on their execution times so that the utilizations of all processors can be initially close to their set points. At time 300×5 seconds, tasks T_6 and T_7 terminate so that the workload becomes uncontrollable. From the experimental results shown in Fig. 7, we can see that only the utilizations of processor P_2 and P_5 converge to the desired set points. The utilization of P_1 stays slightly below the set point. P_4 is severely underutilized as its utilization is just 50 percent. P_3 is overloaded and has a utilization that is much higher than the schedulable bound. The reason for P_3 to become overloaded is that the controller is trying to increase the rates of tasks T_2 and T_3 (as shown in Fig. 6b), so that the utilizations of P_4 and P_5 can converge to the set point 0.7. The utilizations of P_3, P_4, and P_5 (i.e., three control outputs) need to be controlled by actuating only the rates of two tasks (T_2 and T_3, i.e., two control inputs). According to Corollary 4.2, the system becomes uncontrollable. As a result, the controller is unable to control the utilizations of the three processors to converge to their set points by adjusting the rates of T_2 and T_3. The system finally settles with P_4 being underutilized and P_3 being overloaded. As processor overload may cause *deadline misses* as shown in our previous work [37], controllability has to be maintained at runtime.

One may be easily tempted to think that the simple solution of disabling utilization control would work fine here to let processors become underutilized after the task termination. However, as discussed before, a DRE system without control is vulnerable to unexpected workload

TABLE 2
Workload Parameters

Task	*Subtask Allocation*	*Estimated Subtask Execution Time (ms)*				*Initial Rate*	*Min Rate*	*Max Rate*
T_1	P_1, P_2	38	11			23.96	20	60
T_2	P_5, P_4, P_3	22	28	43		29.99	20	60
T_3	P_2, P_3, P_4, P_5	14	20	12	74	19.40	5	30
T_4	P_2, P_1	25	24			12.43	10	60
T_5	P_1, P_2	33	26			26.15	5	75
T_6	P_2, P_3, P_4	26	16	21		16.97	5	20
T_7	P_4, P_1	16	12			44.05	20	60
T_8	P_1, P_5	23	32			10.00	10	60
T_9	P_5, P_1	27	19			10.00	10	60
T_{10}	P_2, P_4	36	29			10.00	10	60

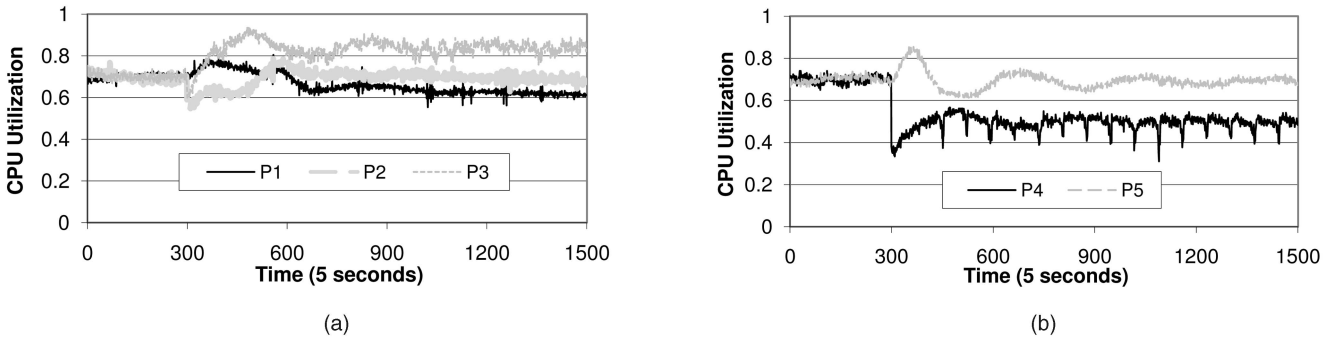

Fig. 7. System becomes uncontrollable after task termination. (a) P1-P3. (b) P4 and P5.

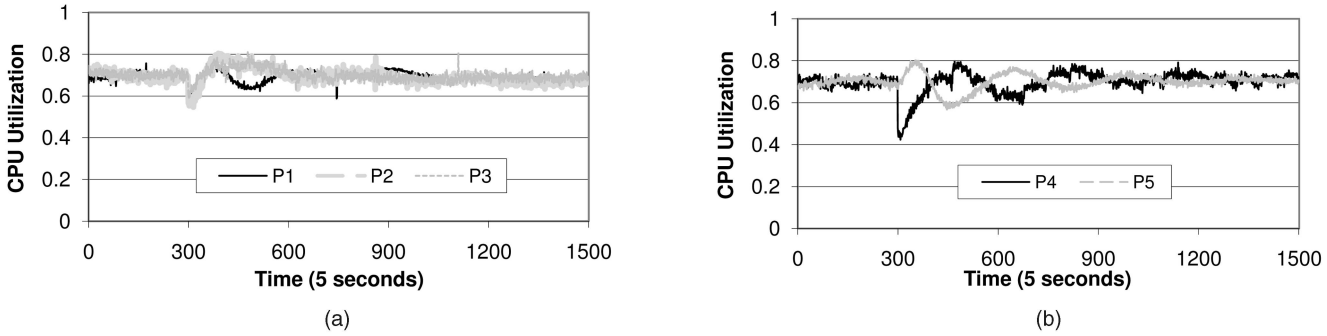

Fig. 8. System becomes controllable after controllability maintenance. (a) P1-P3. (b) P4 and P5.

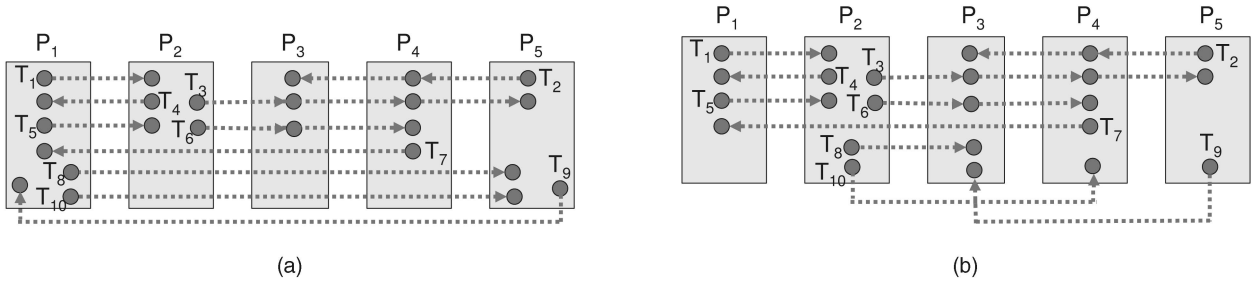

Fig. 9. Workload variations in feasibility experiments. (a) Task allocation after naive solution. (b) Allocation after feasibility adjustment.

variations and load disturbances, which are common for today's DRE systems running in *unpredictable* environments. When variations happen, as shown in our previous work [28], processors in such an open-loop system may soon violate their schedulable utilization bounds, and then, have significant deadline misses. Controlling processors to allow underutilization instead of overload is *not* the best choice either, because tasks running at a higher rate may contribute higher values in some real-time applications. Allowing processors to be underutilized will unnecessarily lower the value of the system [26].

In the second experiment, we run our extended middleware system with the controllability handler activated. All configurations remain the same as in the first experiment. In the controllability analysis, task T_7 is not dedicated to any processor so its termination is ignored. However, task T_6 is dedicated to processor P_4, so we have to migrate a subtask to P_4 after T_6's termination because the two existing subtasks on P_4, T_2, and T_3 are already dedicated to P_3 and P_5, respectively. As an outcome of the online controllability algorithm, subtask $T_{4,2}$ is migrated from processor P_1 to P_4 (as shown in Fig. 6c), immediately after the task terminations. From the results shown in Fig. 8, we can see that the previously uncontrollable system indeed becomes controllable again. The utilizations of all processors converge to the desired set points. Undesired processor overload and underutilization have been avoided. The system value has been increased without causing deadline misses.

8.1.3 Feasibility

As we analyzed before, controllability maintenance alone is not enough because it may still be infeasible for a controllable system to achieve the desired utilization set points when tasks arrive at runtime. In this set of experiments, we first show that some naive allocations of dynamically arriving tasks make it infeasible for the original FC-ORB to achieve the set points. Same as the previous experiments, the utilizations of all processors in the system initially start from their set points. At time 300×5 seconds, three end-to-end tasks (T_8, T_9, and T_{10}) are admitted to the system. As an example of possible naive allocations, three subtasks are allocated to P_1 while the other three are allocated to P_5 (as shown in Fig. 9a). Fig. 10

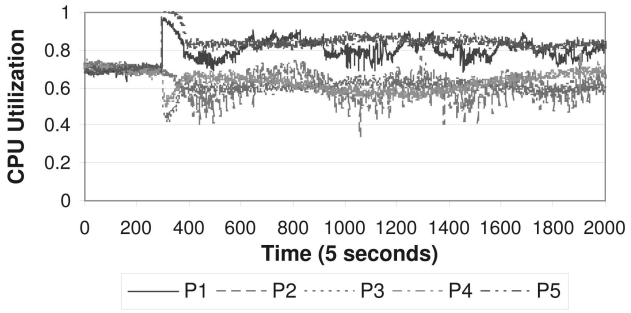

Fig. 10. System becomes infeasible after task arrivals.

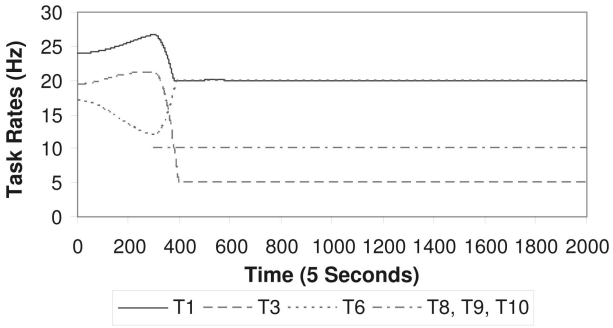

Fig. 11. Task rates saturate at boundaries when the system is infeasible.

TABLE 3
Task Rates of All Tasks (S Means Rate Saturated)

	T_1	T_2	T_3	T_4	T_5
Naive	20 (S)	30.6569	5 (S)	29.9830	5.6836
Feasibility	43.1741	20.6556	19.3107	11.6857	5.0194
	T_6	T_7	T_8	T_9	T_{10}
Naive	20 (S)	50.4092	10 (S)	10 (S)	10 (S)
Feasibility	5.0004	50.5294	10.0018	11.1398	10.0008

shows that the system becomes infeasible after this allocation. P_1 and P_5 become overloaded while P_2-P_4 are underutilized. Fig. 11 and Table 3 show that the rates of several tasks saturate after the task arrivals. The rates of tasks T_1, T_3, and T_8-T_{10} reach their lower boundaries, and so cannot be decreased anymore. On the other hand, the rate of task T_6 reaches the upper boundary, and so cannot be increased any further. As a result of the saturations, no processor can achieve their set points because it is infeasible to do so.

We then run the same experiment on our extended middleware system with the feasibility handler enabled. Whenever new tasks are admitted to the system, the feasibility handler conducts incremental feasibility algorithm presented in Section 6 to allocate the subtasks. We can see that the new tasks first have a smaller impact on the utilizations of the processors in the system, compared to the naive solution. That is because the feasibility handler distributes the impact to different processors, as shown in Fig. 9b. Fig. 12 demonstrates that even though the same task rate constraints exist, the system still can achieve the desired utilization set points because of the feasibility adjustment. As shown in Fig. 13, all task rates that were previously saturated due to system infeasibility no longer saturate. Table 3 shows that none of the tasks saturate at

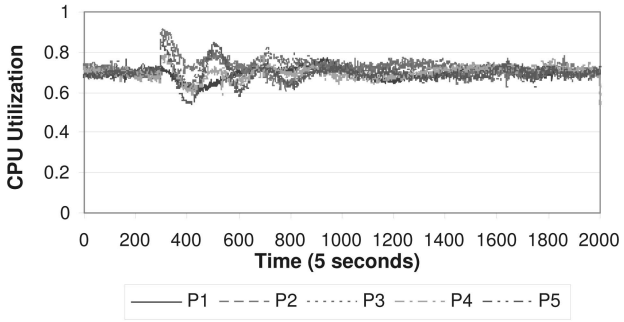

Fig. 12. System remains feasible after feasibility adjustment.

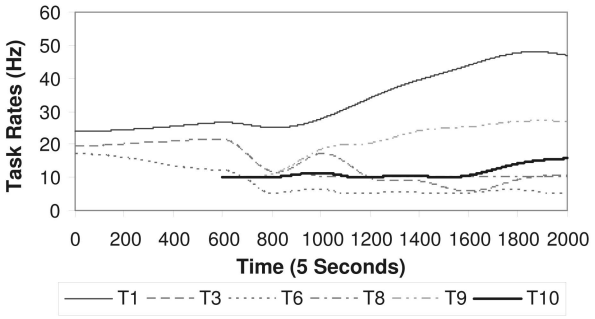

Fig. 13. Task rates no longer saturate after feasibility adjustment.

their rate boundaries. Hence, with feasibility adjustment, it becomes feasible for a previously infeasible system to achieve the desired set points.

8.2 Numerical Results

In this section, we evaluate the offline subtask allocation algorithms using numerical experiments, which allow us to use a large number of randomly generated workloads to stress test the algorithms in large systems.

In all experiments presented in this section, the number of tasks has been fixed at 50 (i.e., $m = 50$), while the number of subtasks of each task varies uniformly from one to seven. For each task, its lower rate bound, $R_{min,j}$, is randomly generated between 0.01 and 0.1 Hz. For each subtask, its minimum estimated utilization varies randomly between 5 and 15 percent and its execution time is calculated based on its rate and its minimum estimated utilization. Each subtask can *only* be executed on five processors (i.e., $p = 5$), which are randomly chosen from all processors, to represent a typical resource constraint. Because any system with more processors than tasks is uncontrollable, we vary the number of processors (i.e., n) from 35 to 50 to examine the performance of our algorithms when the average number of subtasks on each processor changes. For each value of n, 500 different workload configurations are randomly generated and tested. The schedulable utilization bound of RMS [24], namely $B_i = m_i(2^{1/m_i} - 1)$, is used as the utilization set point of each processor P_i to avoid deadline misses, where m_i is the number of subtasks on this processor.

We compare our algorithms against a baseline algorithm called *Simple*, which represents a typical bin-packing-based allocation solution without the consideration of controllability or feasibility. We use Simple as our baseline because,

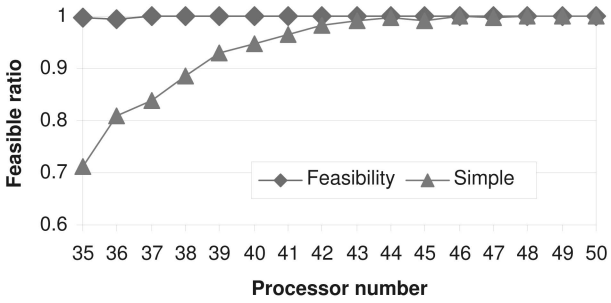

Fig. 14. Feasible ratio under different processor numbers.

to the best of our knowledge, there is *no* existing algorithm that can guarantee controllability and feasibility for DRE systems. Please note that more sophisticated task allocation algorithms would generate similar results as Simple does, as long as they are designed regardless of controllability and feasibility. Simple first ensures there is no *idle* processor by randomly allocating one subtask to each processor because, otherwise, the system is clearly uncontrollable. Then, the rest subtasks are randomly picked and allocated to processors in a round robin manner. Note that Simple is also subject to the utilization and resource constraints. A subtask can *only* be allocated to a processor where the required resource is available. If a processor's utilization bound is violated after being allocated a new subtask, the subtask cannot be allocated to the processor and has to try the next processor. If a subtask cannot be allocated to any processor, the algorithm fails.

We first examine the *feasible ratio* (i.e., the fraction of task allocations that are feasible) under both our feasibility algorithm and Simple. A workload resulted from an allocation is *feasible* if the minimum estimated utilizations of all processors are equal to or lower than their schedulable bounds. Fig. 14 shows that the feasibility algorithm achieves higher feasibility ratio than Simple when the number of processors is smaller than 44. For example, when processor number is 35, more than 30 percent of workloads are not feasible under Simple, but the ratio is only 1 percent under the feasibility algorithm. The reason is that when the number of processors is small, each processor has more subtasks, which decreases the probability for Simple to find feasible solutions.

As discussed in Section 3.3, the main goal of our feasibility algorithm is to increase the feasibility margin. Fig. 15 plots the average feasibility margin of 250 workloads, which are feasible under both the feasibility

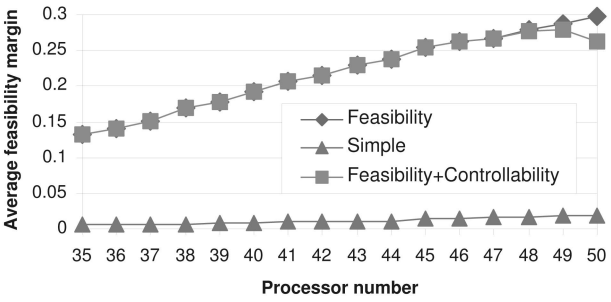

Fig. 15. Feasibility margin under different processor numbers.

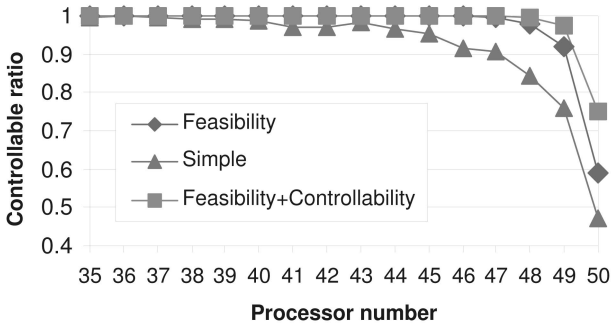

Fig. 16. Controllable ratio under different processor numbers.

algorithm and Simple. The average feasibility margin under Simple is much smaller than that under the feasibility algorithm. That means, the feasibility algorithm results in workloads, which can tolerate much larger execution time variations. For example, with 48 processors, the workload generated by the feasibility algorithm can remain feasible even when task execution times increase by 28 percent. When the number of processors increases, the difference becomes larger. That is because when each processor has fewer subtasks, the space for the feasibility algorithm to improve becomes larger.

We then compare the *controllable ratio* (i.e., the fraction of task allocations that are controllable) under Simple, the feasibility algorithm and the integrated feasibility and controllability algorithm. Same as before, Simple and the feasibility algorithm are applied to all randomly generated workloads without any concern of controllability. In contrast, the integrated algorithm adopts the controllability algorithm introduced in Section 4.2 to reallocate the subtasks if the workload processed by the feasibility algorithm is diagnosed to be uncontrollable. Fig. 16 shows that the controllability algorithm reduces the uncontrollable cases significantly. For example, with 50 processors, the controllable ratio has been increased more than 10 percent. In addition, the feasibility algorithm can also help improve controllability as its controllable ratio is much higher than Simple.

As discussed in Section 5, the controllability algorithm will have some impact on the feasibility margin though the algorithm is designed to minimize the impact. Fig. 15 shows the impact is only roughly 3 percent at a maximum (with 50 processors). This result demonstrates that the controllability algorithm can improve system controllability significantly only at negligible cost of feasibility margin.

9 Conclusion

In this paper, we have shown that both controllability and feasibility are fundamental properties of DRE systems, and so, are crucial to the success of feedback control in such systems. Using end-to-end utilization control as an example, we found that uncontrollable or infeasible DRE systems often cause processor overload, deadline misses, or undesired low task rates. We then proved that controllability and feasibility depend on end-to-end task allocations. We presented both offline and online task allocation

algorithms to ensure system controllability and feasibility both at deployment time and at runtime even when the system is experiencing dynamic workload variations. As a result, a DRE system is guaranteed to meet the end-to-end deadlines of all tasks in the system while being able to run all tasks at the highest possible rates. The resultant high task rates can significantly increase the system value compared to the naive solution of simply keeping processor utilizations under their bounds. Furthermore, we integrated our task allocation algorithms in the FC-ORB middleware. The efficacy of our algorithms has been demonstrated through both empirical results on a physical test bed and numerical evaluations in large systems.

ACKNOWLEDGMENTS

This work was supported in part by the US National Science Foundation (NSF) under CSR Grant CNS-0720663 and US NSF CAREER awards (CNS-0845390, CNS-0448554). The authors would also like to thank the reviewers for their detailed feedback. This is a significantly extended version of a conference paper [36].

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, pp. 74-90, June 2003.

[2] L. Abeni, T. Cucinotta, G. Lipari, L. Marzario, and L. Palopoli, "Adaptive Reservations in a Linux Environment," *Proc. IEEE Real-Time Technology and Applications Symp. (RTAS),* May 2004.

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

[4] S. Ali, J.-K. Kim, Y. Yu, S.B. Gundala, S. Gertphol, H.J. Siegel, and A.A. Maciejewski, "Utilization-Based Techniques for Statically Mapping Heterogeneous Applications onto the HiPer-D Heterogeneous Computing System," *Parallel and Distributed Computing Practices,* 2003.

[5] S. Baruah, "Feasibility Analysis of Preemptive Real-Time Systems upon Heterogeneous Multiprocessor Platforms," *Proc. IEEE Real-Time Systems Symp. (RTSS),* pp. 37-46, 2004.

[6] 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),* Dec. 1998.

[7] M. Brehob, E. Torng, and P. Uthaisombut, "Applying Extra-Resource Analysis to Load Balancing," *Proc. 11th Ann. ACM-SIAM Symp. Discrete Algorithms,* 2000.

[8] M. Caccamo, G. Buttazzo, and L. Sha, "Elastic Feedback Control," *Proc. Euromicro Conf. Real-Time Systems (ECRTS),* 2000.

[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] Y. Diao, J.L. Hellerstein, A.J. Storm, M. Surendra, S. Lightstone, S.S. Parekh, and C. Garcia-Arellano, "Incorporating Cost of Control into the Design of a Load Balancing Controller," *Proc. IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS),* 2004.

[12] G.F. Franklin, J.D. Powell, and M. Workman, *Digital Control of Dynamic Systems,* third ed. Addition-Wesley, 1997.

[13] S. Gertphol, Y. Yu, S.B. Gundala, V.K. Prasanna, S. Ali, J.-K. Kim, A.A. Maciejewski, and H.J. Siegel, "A Metric and Mixed-Integer-Programming-Based Approach for Resource Allocation in Dynamic Real-Time Systems," *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS),* 2002.

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

[15] A. Gokhale, B. Natarajan, D.C. Schmidt, and J.K. Cross, "Towards Real-Time Fault-Tolerant CORBA Middleware," *Cluster Computing: J. Networks, Software, and Applications,* special issue on dependable distributed systems, vol. 7, no. 4, pp. 331-346, 2004.

[16] 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),* May 2004.

[17] C.-J. Hou and K.G. Shin, "Allocation of Periodic Task Modules with Precedence and Deadline Constraints in Distributed Real-Time Systems," *IEEE Trans. Computers,* vol. 46, no. 12, pp. 1338-1356, Dec. 1997.

[18] 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.

[19] C. Karamanolis, M. Karlsson, and X. Zhu, "Designing Controllable Computer Systems," *Proc. USENIX Workshop Hot Topics in Operating Systems (HotOS),* 2005.

[20] 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),* 2005.

[21] C. Lefurgy, X. Wang, and M. Ware, "Server-Level Power Control," *Proc. Fourth IEEE Int'l Conf. Autonomic Computing (ICAC '07),* 2007.

[22] J.P. Lehoczky, "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadline," *Proc. IEEE Real-Time Systems Symp. (RTSS),* 1990.

[23] S. Lin and G. Manimaran, "Double-Loop Feedback-Based Scheduling Approach for Distributed Real-Time Systems," *Proc. High-Performance Computing (HiPC),* 2003.

[24] 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.

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

[26] C. Lu, J.A. Stankovic, G. Tao, and S.H. Son, "Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms," *J. Real-Time Systems,* vol. 23, nos. 1/2, pp. 85-126, July 2002.

[27] 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),* May 2003.

[28] 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.

[29] M.D. Natale and J. Stankovic, "Dynamic End-To-End Guarantees in Distributed Real-Time Systems," *Proc. IEEE Real-Time Systems Symp. (RTSS),* 1994.

[30] 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),* Dec. 1996.

[31] 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),* 2003.

[32] R.W. Shields and J.B. Pearson, "Structural Controllability of Multiinput Linear Systems," *IEEE Trans. Automatic Control,* vol. 21, no. 2, pp. 203-212, Apr. 1976.

[33] 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),* 2001.

[34] D.C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole, "A Feedback-Driven Proportion Allocator for Real-Rate Scheduling," *Proc. USENIX Symp. Operating Systems Design and Implementation,* pp. 145-158, 1999.

[35] J. Sun and J.W.-S. Liu, "Synchronization Protocols in Distributed Real-Time Systems," *Proc. Int'l Conf. Distributed Computing Systems (ICDCS),* 1996.

[36] X. Wang, Y. Chen, C. Lu, and X. Koutsoukos, "On Controllability and Feasibility of Utilization Control in Distributed Real-Time Systems," *Proc. Euromicro Conf. Real-Time Systems (ECRTS),* July 2007.

[37] X. Wang, D. Jia, C. Lu, and X. Koutsoukos, "DEUCON: Decentralized End-To-End Utilization Control for Distributed Real-Time Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 18, no. 7, pp. 996-1009, July 2007.

[38] 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),* 2005.

Xiaorui Wang received the BS degree from the Southeast University, China, in 1995, the M.S. degree from the University of Louisville in 2002, and the PhD degree from Washington University in St. Louis in 2006, all in computer science. In 2005, he was with the IBM Austin Research Laboratory designing power control algorithms for high-performance computing servers. From 1998 to 2001, he was a senior software engineer, and then, a project manager at Huawei Technologies Co. Ltd., China, developing distributed management systems for optical networks. He is an assistant professor in the Department of Electrical Engineering and Computer Science at the University of Tennessee, Knoxville. He is an author or coauthor of more than 30 refereed publications. His research interests include real-time embedded systems, power-aware systems, and cyber-physical systems. He is the recipient of the US National Science Foundation (NSF) CAREER Award in 2009, the Chancellor's Award for Professional Promise in Research and Creative Achievement from the University of Tennessee in 2009, the Power-Aware Computing Award from Microsoft Research in 2008, and the IBM Real-Time Innovation Award in 2007. He received the Best Paper Award at the 29th IEEE Real-Time Systems Symposium (RTSS) in 2008. He is a member of the IEEE and the IEEE Computer Society.

Yingming Chen received the BS degree in computer science from Tsinghua University in 2001 and the MS degree in computer science from Washington University in St. Louis in 2007. He is currently a software engineer at Microsoft Corp.

Chenyang Lu received the BS degree from the University of Science and Technology of China in 1995, the MS degree from Chinese Academy of Sciences in 1997, and the PhD degree from the University of Virginia in 2001, all in computer science. He is an associate professor of computer science and engineering at Washington University in St. Louis. He is the author or coauthor of more than 80 publications, and received an US National Science Foundation (NSF) CAREER Award in 2005 and a Best Paper Award at International Conference on Distributed Computing in Sensor Systems in 2006. He is an associate editor of the *ACM Transactions on Sensor Networks* and the *International Journal of Sensor Networks*, and guest editor of the Special Issue on Real-Time Wireless Sensor Networks of the *Real-Time Systems Journal*. He also served as a general chair and program chair of the IEEE Real-Time and Embedded Technology and Applications Symposium in 2009 and 2008, respectively, demo chair of the ACM Conference on Embedded Networked Sensor Systems in 2005, and track chair of the IEEE Real-Time Systems Symposium in 2007. His research interests include real-time embedded systems, wireless sensor networks, and cyber-physical systems. He is a member of the ACM and the IEEE.

Xenofon D. Koutsoukos received the diploma in electrical and computer engineering from the National Technical University of Athens, Greece, in 1993, the MS degrees in electrical engineering and applied mathematics, and the PhD degree in electrical engineering from the University of Notre Dame, Notre Dame, Indiana, in 1998 and 2000, respectively. From 2000 to 2002, he was a member of research staff with the Xerox Palo Alto Research Center, Palo Alto, California, working in the Embedded Collaborative Computing Area. Since 2002, he has been with the Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN, where he is currently an assistant professor and a senior research scientist at the Institute for Software Integrated Systems. His research interests include hybrid systems, real-time embedded systems, sensor networks, and cyber-physical systems. He currently serves as an associate editor for the *ACM Transactions on Sensor Networks* and for *Modelling Simulation Practice and Theory*. He is a senior member of the IEEE and a member of the ACM. He was the recipient of the US National Science Foundation CAREER Award in 2004.