Real-Time Syst (2013) 49:339–366
DOI 10.1007/s11241-012-9168-7

Optimal and efficient adaptation in distributed real-time systems with discrete rates

Yingming Chen · Chenyang Lu · Xenofon D. Koutsoukos

Published online: 22 November 2012

Abstract Many distributed real-time systems face the challenge of dynamically maximizing system utility and meeting stringent resource constraints in response to fluctuations in system workload. Thus, online adaptation must be adopted in face of workload changes in such systems. We present the *MultiParametric Rate Adaptation* (*MPRA*) algorithm for discrete rate adaptation in distributed real-time systems with end-to-end tasks. The key novelty and advantage of MPRA is that it can *efficiently* produce *optimal* solutions in response to workload variations caused by dynamic task arrivals and departures. Through offline preprocessing MPRA transforms an NP-hard utility optimization problem to the evaluation of a piecewise linear function of the CPU utilization. At run time MPRA produces optimal solutions by evaluating the function based on the CPU utilization. Analysis and simulation results show that MPRA maximizes system utility in the presence of varying workloads, while reducing the online computation complexity to polynomial time. The advantages of MPRA have been validated through the implementation in a real-time middleware system and experiments on a physical testbed.

Keywords Real-time systems · Middleware · Rate adaptation

Y. Chen · C. Lu (✉)
Department of Computer Science and Engineering, Washington University in St. Louis, One Brookings Drive, St. Louis, MO 63130, USA
e-mail: lu@cse.wustl.edu

Y. Chen
e-mail: yingming@cse.wustl.edu

X.D. Koutsoukos
Department of EECS, Vanderbilt University, Box 1679, Station B, Nashville, TN 37235, USA
e-mail: Xenofon.Koutsoukos@vanderbilt.edu

1 Introduction

An increasing number of distributed real-time systems operate in dynamic environments where system workload may change at run time (Abdelzaher et al. 2003). A key challenge faced by such systems is to dynamically maximize system utility subject to resource constraints and fluctuating workload. For instance, the Supervisory Control and Data Acquisition (SCADA) system of a power grid may experience dramatic load increase during cascading power failures and cyber attacks. Similarly, the arrival rate of service requests in an online trading server can fluctuate dramatically. However, such systems must meet stringent resource constraints despite their fluctuating workload. Online adaptation must be adopted to handle workload changes in such systems.

Online adaptation introduces several important challenges in distributed real-time systems. First, online adaptation should *maximize system utility* subject to multiple resource constraints. For example, many distributed real-time systems must enforce certain CPU utilization bounds on multiple processors in order to prevent system crash due to CPU saturation and meet end-to-end deadlines (Wang et al. 2005b). Second, many common adaptation strategies only support *discrete* options. For example, an admission controller must make binary decisions (admission/rejection) on a task. While task rate adaptation can allow a system to adapt at a finer granularity (Buttazzo et al. 2002; Cervin et al. 2002; Lu et al. 2002, 2005; Koutsoukos et al. 2005; Steere et al. 1999), many real-time applications (e.g., avionics (Abdelzaher et al. 2000) and Multiple Bit-Rate Video) can only run at a discrete set of predefined rates. Unfortunately, utility optimization problems with discrete options are NP-hard (Lee et al. 1999b). Furthermore, despite the difficulty of such problems, a real-time system must adapt to dynamic workload changes quickly, which requires optimization algorithms to be highly efficient at run time.

Existing approaches to utility optimization in real-time systems can be divided into two categories: optimal solutions and efficient heuristics. Approaches based on integer programming or dynamic programming have been proposed to optimize utility (Lee et al. 1999a, 1999b). While these approaches produce optimal solutions, they are computationally expensive and cannot be used *online*. On the other hand, a number of efficient heuristics have been proposed for online adaptation (Rajkumar et al. 1998; Lee et al. 1999b; Abdelzaher et al. 2000; Lee et al. 2004). However, these algorithms can only produce suboptimal solutions in terms of system utility.

To overcome the limitations of existing approaches, we present the *MultiParametric Rate Adaptation* (*MPRA*) algorithm for online adaptation in real-time systems. MPRA employs admission control or task rate adaptation as the online adaptation mechanism, which is supported by a broad range of real-time applications, such as digital control (Cervin et al. 2002), video streaming, and avionics (Abdelzaher et al. 2000). While several rate adaptation algorithms (Lu et al. 2005; Wang et al. 2005a; Koutsoukos et al. 2005; Abdelzaher et al. 2000) have been proposed for distributed real-time systems, some of them (Koutsoukos et al. 2005; Abdelzaher et al. 2000) are heuristics-based suboptimal solutions, while others can only deal with *continuous* rates. Unfortunately, many real-time applications can only support a finite set of *discrete* rates. For example, in a sensor-to-weapon shooter system, changing the task rates implies changing the rate at which sensed imagery data are published. Setting the task rates to any value within a range is not practical since the hardware that senses the data may not have a high-resolution timer needed to precisely program the tasks. In a fully-automated flight control system, each flight task only accept a spectrum of Quality-of-Service (QoS) levels with each level associated to a concrete task rate (Abdelzaher et al. 2000). Some multimedia applications (e.g., Multi Bit Rate video) also only support a few predefined rates. Due to its computational complexity, optimal discrete rate adaptation is particularly challenging in distributed real-time systems which requires fast adaptation in response to workload changes. To meet this challenge, MPRA is designed to handle a common class of end-to-end tasks that may only execute at a *discrete* set of rates on *multiple* processors. Online adaptation in such systems represents a particularly challenging problem as it is NP-hard (as discussed in Sect. 3).

The key novelty and advantage of our approach is that it can *efficiently* produce *optimal* solutions online in face of dynamic task arrivals and departures. The MPRA algorithm is based on *multiparametric mixed-integer linear programming (mp-MILP)* (Acevedo and Pistikopoulos 1999). Through offline preprocessing MPRA transforms an NP-hard utility optimization problem to the evaluation of a piecewise linear function of the CPU utilization. At run time MPRA produces optimal solutions by evaluating the function based on the workload variation. Specifically, the primary contributions of this paper are four-fold:

– We present MPRA, a novel algorithm for discrete rate adaptation in distributed real-time systems with end-to-end tasks;
– We provide analysis that proves that our algorithm produces optimal system utility in face of workload changes with the online rate adaptation running in *polynomial* time;
– We present simulation results that demonstrate that MPRA maximizes system utility in the presence of dynamic task arrivals, with the online execution time comparable to efficient suboptimal heuristics and two orders of magnitude lower than a representative optimal solver.
– We integrate our algorithm with the FC-ORB real-time middleware (Wang et al. 2005b) and report empirical results on an experimental testbed.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 formalizes the optimization problem addressed in this paper. Section 4 presents the design and analysis of our algorithm. Section 5 provides simulation results. Section 6 introduces the implementation of the algorithm in a real-time middleware system and presents empirical results. Finally, Sect. 7 concludes this paper.

2 Related work

Several projects investigated the problem of maximizing system utility in real-time systems. Rajkumar et al. proposed the QoS-based Resource Allocation Model (Q-RAM) (Rajkumar et al. 1997) for utility optimization in distributed real-time systems. Lee et al. presented several optimal algorithms for the Q-RAM model based on integer programming and dynamic programming (Lee et al. 1999a, 1999b). These approaches are computationally expensive and unsuitable for online adaptation in real-time systems. To improve the efficiency of the solutions, the authors also proposed several efficient heuristic algorithms that can only produce suboptimal solutions (Rajkumar et al. 1998; Lee et al. 1999a, 1999b; Lee and Siewiorek 1998; Ghosh et al. 2003). Specifically, they presented heuristic algorithms with bounded approximation ratio for the single-resource case (Lee et al. 1999a; Lee and Siewiorek 1998). However, the heuristic algorithms for multi-resource problems do not have analytical bounds on the approximation ratio (Lee and Siewiorek 1998). Note that the multi-resource case is common in distributed real-time systems in which each processor is a separate resource.

Several middleware systems have been developed to improve system utility by dynamically adjusting the QoS levels of applications. The authors in Tokuda and Kitayama (1994), Brandt et al. (1998), Brandt and Nutt (2002) have developed middleware solutions that support mediating application resource usage using application QoS levels for single processor systems. Abdelzaher et al. developed a QoS-negotiation model and incorporated it into an example real-time middleware service, called RTPOOL, in Abdelzaher et al. (2000). However, all the above middleware systems employ heuristic algorithms that cannot produce optimal solutions.

Recently, Lee et al. introduced a method called service class configuration to address the online adaptation problem with dynamic arrival and departure of tasks in surveillance radar systems (Lee et al. 2004). This method avoids running optimization procedures at run time by designing a set of service classes offline, which will be used adaptively depending on the system state. While service classes can effectively improve the efficiency of online adaptation, it cannot produce optimal solutions. In contrast, MPRA can produce optimal solutions with efficient online execution.

Several task rate adaptation algorithms have been proposed for single-processor (Cervin et al. 2002; Buttazzo et al. 2002; Steere et al. 1999) and distributed real-time systems (Lu et al. 2005; Wang et al. 2005a). All the above solutions assume that task rates can be adjusted in a *continuous* range. As discussed in Sect. 1, this assumption does not hold in many applications that only support *discrete* configurations. HySUCON (Koutsoukos et al. 2005) is a heuristic algorithm for real-time systems that supports discrete task rates. However, it is designed for single processor systems and cannot produce optimal solutions. There are two important differences between our work and earlier work on rate adaptation. First, our work deals with real-time systems with discrete task rates, while none of the aforementioned rate adaptation algorithms (with the exception of Rajkumar et al. 1997) is designed to handle discrete rates. Moreover, none of them can maximize system utility.

There are also several methods developed to handle rate adaptation for specific applications, such as control systems, power management, security and thermal control (Wang et al. 2009; Chen et al. 2010; Fu et al. 2010a, 2010b; Lindberg and Årzén 2010). However, like those algorithms for single-processor, these methods all assume continuously adjustable task rates, which are not supported in those applications only with *discrete* task rates.

This paper is an extension to an earlier conference paper (Chen et al. 2007). While Chen et al. (2007) only includes the theoretical design and simulations of the algorithm, we present the implementation of the algorithm in a real distributed middleware system and empirical evaluation of the system on a physical testbed in this paper. In addition, this paper presents the details of the transformation from the end-to-end admission control problem to an mp-MILP problem as well as the corresponding simulation results and analysis that compare MPRA with three baseline algorithms.

3 Problem formulation

We now formulate the discrete rate adaptation problem in distributed real-time systems.

3.1 End-to-end task model

We classify tasks in distributed real-time systems into two categories: *adaptable* tasks and *unadaptable* tasks. Tasks that may be rejected or support multiple rates are called *adaptable* tasks; tasks that must be admitted and executed at fixed rates are called *unadaptable* tasks; tasks with fixed rates fall into *unadaptable* task category. Mission critical tasks that must execute at fixed rates are typical *unadaptable* tasks.

The system is comprised of m *adaptable* periodic tasks $\{T_i | 1 \le i \le m\}$ executing on n processors $\{P_i | 1 \le i \le n\}$. Task T_i is composed of a graph of subtasks $\{T_{ij} | 1 \le j \le m_i\}$ that may be located on different processors. We denote the set of subtasks of *adaptable* task T_i that are allocated on P_j as S_{ji}. Due to the dependencies among subtasks each subtask T_{ij} of a periodic task T_i shares the same rate as T_i.[1] Each task T_i is subject to an end-to-end relative deadline related to its period τ_i. Each subtask T_{ij} has an execution time c_{ij}.

Each *adaptable* task only supports a set of discrete task rates for online adaptation. A task running at a higher rate contributes a higher utility to the system at the cost of higher utilization. We denote the set of discrete rate choices of task T_i as $R_i = \{r_i^{(0)}, \dots, r_i^{(k_i)}\}$ in increasing order. The set of utility options for task T_i is denoted by $Q_i = \{q_i^{(0)}, \dots, q_i^{(k_i)}\}$ where $q_i^{(j)}$ is the utility value contributed by T_i when it is executed at rate $r_i^{(j)}$. Note that we do not make any assumption regarding the relation between the utility of the task and the task rate. For example, a task's utility values do not need to be a linear or polynomial function of the task rate. MPRA can handle arbitrary utility values assigned to discrete task rates. Task utility values for different rates can be represented by a lookup table, which is specified by application designers based on domain knowledge. Admission control is a special case of discrete rate adaptation, in which each task only has two rate choices: zero when the task is evicted and a fixed non-zero rate when task is admitted.

3.2 Discrete rate adaption problem

Before formulating the discrete rate adaptation problem, we first introduce several notations:

[1] A non-greedy synchronization protocol (Sun and Liu 1996) can be used to remove release jitter of subtasks.

- $R=[r_1,\dots,r_m]$ is the task rate vector where r_i is the current invocation rate of task T_i. $r_i \in R_i$, $1 \le i \le m$.
- Q_s is the system utility, i.e., the combined utility of all *adaptable* tasks defined as the weighted sum of the task utilities $Q_s=\sum_{i=1}^{m} w_i q_i$ where q_i, $q_i \in Q_i$, is the current task utility of T_i and $0 \le w_i \le 1$, $1 \le i \le m$, are weights describing the relative importance of the tasks. Task weights are defined by the user and are independent from other variables such as task priority. The weight represents the importance of a task relative to the others in the system. If a task has a larger weight, it is more beneficial to increase the utility achieved by this task. Note that we do not consider the utility of *unadaptable* tasks as they must be executed at fixed rates and hence contribute fixed utility.
- $D=[d_1,\dots,d_n]$ is the workload variation vector where d_i is the change to the utilization of the ith processor caused by dynamic arrivals or departures of *unadaptable* tasks with fixed rates. D can be calculated based on the worst case execution times and rates of the *unadaptable* tasks that are assumed to be known. For example, denote the set of *unadaptable* tasks that arrive as S_a and the set of *unadaptable* tasks that depart as S_b. Then $d_i=\sum_{T_j\in S_a}\sum_{T_{jl}\in S_{ij}} c_{jl} r_j - \sum_{T_j\in S_b}\sum_{T_{jl}\in S_{ij}} c_{jl} r_j$ where S_{ij} is the set of subtasks of T_j that run on processor P_i, c_{jl} is the worst-case execution time of subtask T_{jl}, and r_j is the current rate of T_j.
- $U=[u_1,\dots,u_n]$ is the CPU utilization vector where u_i represents the utilization of the ith processor, i.e., the average fraction of time when the ith processor is not idle. Note that u_i includes the utilizations of both *adaptable* and *unadaptable* tasks, i.e., $u_i=d_i+\sum_{1\le j\le m}\sum_{T_{jl}\in S_{ij}} c_{jl} r_j$.
- $B=[b_1,\dots,b_n]$ is the utilization bound vector where b_i is the utilization bound of the ith processor specified by user.

The discrete rate adaptation problem can be formulated as a constrained optimization problem. The goal is to maximize the system utility by selecting the rates of *adaptable* tasks in response to workload changes when *unadaptable* tasks dynamically arrive or depart, i.e.

$$\max_R \sum_{i=1}^{m} w_i q_i \tag{1}$$

subject to

$$r_i \in R_i, \quad 1 \le i \le m \tag{2}$$

$$U \le B \tag{3}$$

The constraint (2) indicates that each task can only be configured with predefined rates. The utilization constraint (3) is used to enforce certain CPU utilization bounds on multiple processors in order to meet the following two goals:

- *Meeting end-to-end deadlines*. Real-time tasks must meet their end-to-end deadlines in distributed real-time systems. In the end-to-end scheduling approach (Sun and Liu 1996), the deadline of an end-to-end task is divided into subdeadlines of its subtasks, and 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 for meeting the subdeadlines on a processor is by enforcing the schedulable utilization bound (Liu and Layland 1973; Lehoczky 1990). The subdeadlines of all the subtasks on a processor are guaranteed if the utilization of the processor remains below its schedulable utilization bound. To meet end-to-end deadlines, the utilization set point of each processor is set to a value below its schedulable utilization bound. We can apply various subdeadline assignment algorithms (Kao and Garcia-Molina 1997; Natale and Stankovic 1994) and schedulable utilization bounds for different task models (Liu and Layland 1973; Lehoczky 1990) presented in the literature.

– *Overload protection*. Many systems must avoid saturation of CPUs, which may cause system crash or severe service degradation (Abdelzaher et al. 2002). On COTS operating systems that support real-time priorities, high utilization by real-time threads may cause kernel starvation (Lu et al. 2003). The utilization constraint (3) allows a user to enforce desired utilization bounds for the processors in a distributed system. Note that overload protection is important for both real-time and many non-real-time distributed systems.

The discrete rate adaptation problem is NP-hard as it can be easily reduced to the 0-1 Knapsack Problem (Martello and Toth 1990). It is therefore impractical to apply standard optimization approaches to discrete rate adaptation in distributed real-time systems. There exist several approximation algorithms for the 0-1 Knapsack Problem that run in polynomial time (Ibarra and Kim 1975; Sahni 1975). However, those algorithms can only handle problems for the single-resource case and can not be applied for multi-resource problems.

4 Design and analysis of MPRA

In this section, we present the design and analysis of MPRA. The MPRA algorithm is based on *Multiparametric programming*, which is a general framework for solving mathematical programming problems with constraints that depend on varying parameters (Gal and Nedoma 1972). This technique is suitable for discrete rate adaptation problems, where the utilization constraints are related to online workload variations. In the rest of this section, we first give a brief overview of the general framework of multiparametric programming. Next, we transform the discrete rate adaptation problem to multiparametric programming formulation and design MPRA for optimal and efficient rate adaptation in distributed real-time systems. Finally, we present the complexity analysis of our algorithm.

4.1 Multiparametric programming

Multiparametric programming provides a systematic way to analyze the effect of parameter changes on the optimal solution of a mathematical programming problem. Rather than solving the optimization problem completely online, it includes an offline and an online step. The offline algorithm partitions the space of varying parameters into regions. For each region, the objective and optimization variables are expressed as linear functions of the varying parameters. For a given value of the varying parameter, the online algorithm computes the optimal solution by evaluating the function for the region which includes the parameter value.

The multiparametric approach has been extended for multiparametric mixed-integer linear programming problems (mp-MILP) (Acevedo and Pistikopoulos 1999). The algorithm presented in Acevedo and Pistikopoulos (1999) uses a Branch and Bound strategy to solve multi-parametric 0-1 mixed-integer linear programming problems of the following form:

$$\min_{x} z(\theta) = cx \tag{4}$$

subject to

$$Ax \le b + F\theta \tag{5}$$

$$G\theta \le g \tag{6}$$

$$\theta \in \Re^s \tag{7}$$

where the elements of the optimization vector x can be either continuous or binary variables, and the vector θ is a vector of parameters varying in $\Xi = \{\theta | G\theta \le g; \theta \in \Re^s\}$. The optimal solution to the problem is a set of linear functions of parameters where each function is corresponding to one region of the parameter space. By combining all linear functions together the optimal solution is a piecewise affine (PWA) function with a polyhedral partition of the following form

$$x(\theta) = P_i\theta + q_i, \quad \text{if } H_i\theta \le k_i, i = 1, \ldots, N_r \tag{8}$$

where the regions $\Theta_i \overset{\triangle}{=} \{\theta \in \Xi : H_i\theta \le k_i\}$, $i = 1, \ldots, N_r$ form a partition of the entire space of varying parameters. The optimality of the mp-MILP approach is ensured by exhaustiveness, as in any standard Branch and Bound algorithm.

We observe the mp-MILP approach is suitable for real-time systems that must handle workload changes by switching among discrete rates. The key advantage of the multiparametric programming is that, while the offline step may have a high time complexity, the online step can generate optimal solutions efficiently. As a result, the optimal solution can be computed quickly in response to workload changes. This characteristic makes it very suitable for the discrete rate adaptation problem in dynamic distributed real-time systems, which require both optimal resource allocation and fast response to workload variations. To our knowledge MPRA is the first instantiation of the general multiparametric programming approach in real-time systems.

4.2 Problem transformation

The key step in the design of MPRA is to transform the discrete rate adaptation problem presented in Sect. 3.2 to an mp-MILP problem, after which the mp-MILP approach is adopted to solve the problem. We start with the transformation of the admission control problem, which is a special case of discrete rate adaptation, followed by the general case.

4.2.1 End-to-end admission control

In admission control, each task T_i only has two rate choices: $r_i^{(0)}$ ($r_i^{(0)} = 0$, i.e., T_i is evicted) and $r_i^{(1)}$ ($r_i^{(1)} > 0$, i.e., T_i is admitted). We focus on admission control first because it allows a simpler transformation than the general rate adaptation problem. We introduce an admission vector X with m elements to represent rate choices for all *adaptable* tasks such that

$$x_i = \begin{cases} 1 & \text{if } T_i \text{ is admitted} \\ 0 & \text{if } T_i \text{ is evicted} \end{cases} \tag{9}$$

We introduce an $n \times m$ matrix F, where $f_{ij} = \sum_{T_{jl} \in S_{ij}} r_j^{(1)} c_{jl}$, i.e., the total utilization of task T_j's subtasks on processor P_i if T_j is admitted, and $f_{ij} = 0$ if no subtask of T_j is allocated on processor P_i. The CPU utilization vector U follows the following relationship with the workload variation vector D and the admission vector X:

$$U = FX + D \tag{10}$$

If we assume the task utility contributed by T_i is zero when it is evicted, i.e., $q_i^{(0)} = 0$, then the task utility of T_i can be obtained by $q_i^{(1)} x_i$ where $q_i^{(1)}$ is the task utility contributed by T_i when it is admitted. We introduce a vector Q such that $q_i = w_i q_i^{(1)}$, $1 \le i \le m$. Thus, the system utility can be obtained by $Q_s = QX$. By denoting $D_N = B - D$, we transform this admission control problem to the following mp-MILP problem with D_N as the varying parameter:

$$\min_X (-QX) \tag{11}$$

subject to

$$FX \le D_N \tag{12}$$

$$x_i \in \{0, 1\}, \quad 1 \le i \le m \tag{13}$$

The constraint (12) enforces the CPU utilization bounds specified by the user on all processors. The constraint (13) indicates that each task supports only a set of discrete task rates.

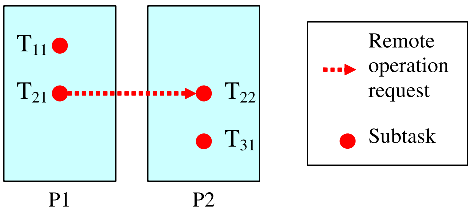


Fig. 1 An example workload

Example Suppose there are two processors and three adaptable tasks in the system. As shown in Fig. 1, T_1 has only one subtask T_{11} on processor P_1. T_2 has two subtasks T_{21} and T_{22} on processors P_1 and P_2, respectively. T_3 has one subtask T_{31} allocated to processors P_2. After the problem transformation, we have

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \qquad F = \begin{bmatrix} r_1^{(1)} c_{11} & r_2^{(1)} c_{21} & 0 \\ 0 & r_2^{(1)} c_{22} & r_3^{(1)} c_{31} \end{bmatrix},$$

$$Q = \begin{bmatrix} w_1 q_1^{(1)} & w_2 q_2^{(1)} & w_3 q_3^{(1)} \end{bmatrix}, \qquad D_N = \begin{bmatrix} b_1 - d_1 \\ b_2 - d_2 \end{bmatrix}.$$

4.2.2 Discrete rate adaption

We first introduce a rate adaptation vector X with $\bar{m}$ elements, where $\bar{m} = \sum_{1 \le i \le m} k_i$ and k_i is the number of non-zero rate choices of adaptable task T_i, to represent the rate configuration of the system such that

$$x_l = \begin{cases} 1 & \text{if } T_i \text{ is configured with } r_i^{(j)} \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

where $l = \sum_{1 \le s < i} k_s + j$, $1 \le i \le m$, and $1 \le j \le k_i$. Each 0-1 element in X corresponds to one non-zero rate choice of some tasks in an appropriate order. The task rate vector R can be obtained by $R = ZX$, where Z is an $m \times \bar{m}$ matrix such that

$$z_{il} = \begin{cases} r_i^{(j)} & \text{if } \sum_{1 \le s < i} k_s < l \le \sum_{1 \le s \le i} k_s \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

where $1 \le i \le m$, $1 \le l \le \bar{m}$, and $j = l - \sum_{1 \le s < i} k_s$. Each row in Z is associated with one adaptable task and contains the information of the non-zero rate options for the task.

We then introduce an $n \times m$ matrix H, where $h_{ij} = \sum_{T_{jl} \in S_{ij}} c_{jl}$, i.e., the total execution time of task T_j's subtasks on processor P_i, and $h_{ij} = 0$ if no subtask of T_j is allocated on processor P_i. The model that characterizes the relationship between U and X is given by

$$U = HZX + D \tag{16}$$

To describe the relationship between Q_s and X, we introduce a vector $\bar{Q}$ such that $\bar{q}_l = w_i q_i^{(j)}$ where $l = \sum_{1 \le s < i} k_s + j$, $1 \le i \le m$, and $1 \le j \le k_i$. Each element in $\bar{Q}$ corresponds to one non-zero rate choice of some task. Thus, the system utility is calculated by $Q_s = \bar{Q}X$. By denoting $D_N = B - D$ and $G = HZ$, we re-formulate the discrete rate adaptation problem as following:

$$\min_X (-\bar{Q}X) \tag{17}$$

subject to

$$GX \le D_N \tag{18}$$

$$x_i \in \{0, 1\}, \quad 1 \le i \le \bar{m} \tag{19}$$

$$\sum_{\sum_{1\le s<i} k_s < j \le \sum_{1\le s\le i} k_s} x_j \le 1, \quad 1 \le i \le m \tag{20}$$

The constraint (18) enforces desired CPU utilization bounds on all processors. The constraint (19) requires that each task only supports a finite set of task rate choices. For each task only one rate choice can be selected at a time, which is ensured by the constraint (20).

Considering D_N as the varying parameter vector and X as the optimization vector, we have transformed the discrete rate adaptation problem to an mp-MILP problem.

Example We still use the example workload shown in Fig. 1 to demonstrate how to formulate a discrete rate adaptation problem. In this example each adaptable task has two non-zero rate options, and hence $\bar{m} = 6$. We have

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix}, \qquad Z = \begin{bmatrix} r_1^{(1)} & r_1^{(2)} & 0 & 0 & 0 & 0 \\ 0 & 0 & r_2^{(1)} & r_2^{(2)} & 0 & 0 \\ 0 & 0 & 0 & 0 & r_3^{(1)} & r_3^{(2)} \end{bmatrix},$$

$$H = \begin{bmatrix} c_{11} & c_{21} & 0 \\ 0 & c_{22} & c_{31} \end{bmatrix}, \qquad D_N = \begin{bmatrix} b_1 - d_1 \\ b_2 - d_2 \end{bmatrix},$$

$$Q = \begin{bmatrix} w_1 q_1^{(1)} & w_1 q_1^{(2)} & w_2 q_2^{(1)} & w_2 q_2^{(2)} & w_3 q_3^{(1)} & w_3 q_3^{(2)} \end{bmatrix}.$$

The constraint (20) can be described by

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} X \le \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

4.3 Design of MPRA

After transforming the discrete rate adaptation problem to an mp-MILP problem, we present the MPRA algorithm that can dynamically select *optimal* rates for adaptable tasks in response to workload changes such as dynamic arrival and departure of un-adaptable tasks. As shown in Fig. 2, MPRA has both an offline part and an online part. In the following, we present the functionality of each component in detail.

4.3.1 Offline components

The offline part of MPRA including an mp-MILP Solver and a Search Tree Generator only executes once before the system starts running. It first invokes the mp-MILP Solver to generate the piecewise affine (PWA) function and then calls the Search Tree Generator to build a binary tree that represents the PWA function.

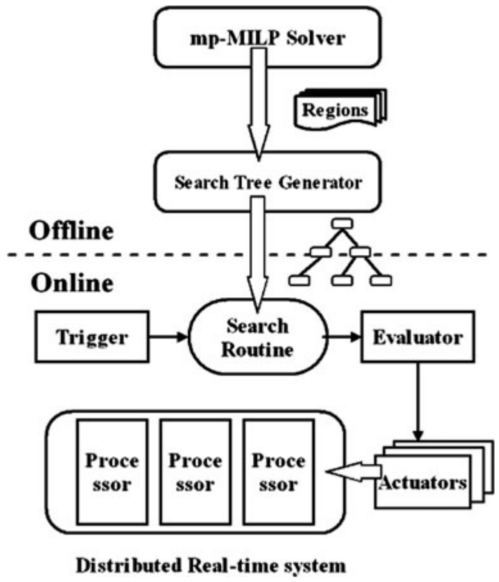


Fig. 2 Overview of MPRA

mp-MILP Solver MPRA invokes the mp-MILP Solver to divide the n-dimensional space of D_N into multiple regions and generates the PWA function which expresses X as a linear function of D_N for each region. The mp-MILP Solver implements a Branch and Bound algorithm that recursively fixes the 0-1 variables in X and builds an enumeration tree to generate the PWA function. Each node in the tree corresponds to an intermediate mp-MILP problem with all remaining 0-1 variables. The space of D_N to be considered for this intermediate problem is defined as the set of regions found for the parent node. At each node, an mpLP problem is solved by relaxing the 0-1 variables as continuous variables in [0,1]. The solution of a non-leaf node is a lower bound of any integer solution to the intermediate mp-MILP problem. The solution of a leaf node, where all 0-1 variables have been fixed, is an *integer* solution of the final mp-MILP problem in a set of regions. At any level of the tree, the current solution is compared with the upper bound to eliminate parts of the space of D_N defined for the remaining nodes. Note that the integer solution at each leaf node is feasible (i.e., meets the utilization constraints), but may not be optimal for the final mp-MILP problem in terms of system utility, because the regions for different leaf nodes can overlap with each other. This is undesirable because, for some given value of D_N that belongs to the intersection of multiple regions, the online part would have to compare the solutions in all those regions to find the optimal one. To facilitate efficient online calculation, the Solver removes the overlap among the regions for all leaf nodes by dividing them into non-overlapping subregions each corresponding to the optimal solution.

Example Recall the example workload shown in Fig. 1 to demonstrate how the offline part of MPRA works. For clarity of presentation we consider the admission control case, i.e., each adaptable task only has one nonzero rate choice. Task parameters are given in Table 1. The parameter space is a 2-dimensional box, $0 \le D_N(1) \le 2$ and

Table 1 Task parameter in the example workload

T_{ij}	c_{ij}	$r_i^{(1)}$	$q_i^{(1)}$	ω_i
T_{11}	20	1/50	0.6	1
T_{21}	20	1/80	1.0	1
T_{22}	20			
T_{31}	45	1/100	0.8	1

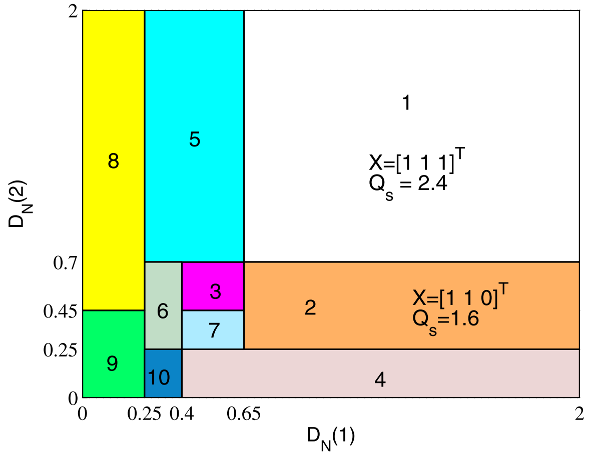


Fig. 3 Partition of the parameter space with the example workload

$0 \le D_N(2) \le 2$, for this concrete problem. The mp-MILP Solver divides the entire space into 10 regions (see Fig. 3). Each region corresponds to one optimal integer solution. For example, $X = [1\ 1\ 0]^T$ for region 2, i.e., for any given value of D_N in region 2, only T_1 and T_2 will be admitted in order to maximize the utility while meeting the utilization constraints. We observe that all regions generated by the offline part of MPRA are rectangles. This is because the coefficients of D_N are identity matrices in constraints (12) and (18).

Search tree generator The search tree generator generates a binary tree data structure that represents the PWA function generated by the mp-MILP Solver. Each node in the tree corresponds to a polyhedron which consists of a set of regions generated by the mp-MILP Solver. An intermediate node contains the inequality for one selected hyperplane that is best for balancing the node's left and right child in terms of the number of linear functions. Each leaf node corresponds to one unique linear function that can be evaluated to obtain the optimal solution for any given value of D_N that belongs to the polyhedron corresponding to this node. For a given D_N the online part only evaluates one linear inequality at each level and then selects the left or right sub-tree to continue based on the sign. With the help of the binary tree, the time of the online evaluation of the PWA function becomes *logarithmic* in the number of regions.

We implemented the offline part of MPRA using the MPT toolbox (Kvasnica et al. 2004), which provides an mp-MILP solver (Acevedo and Pistikopoulos 1999) and a binary tree generator (Tondel et al. 2003).

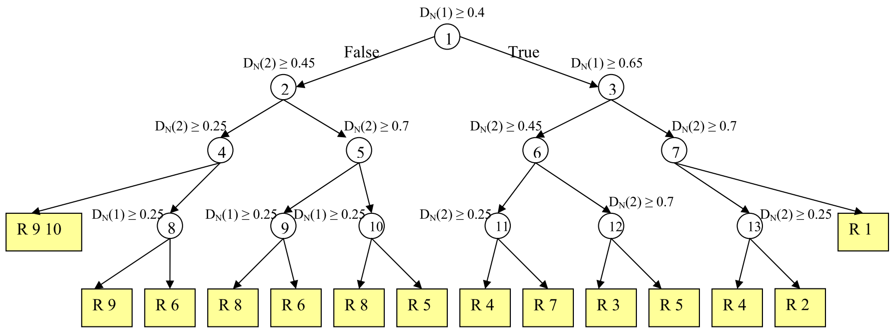


Fig. 4 Binary tree generated from the example partition shown in Fig. 3

Example Figure 4 shows the binary tree generated by the Search Tree Generator from the example space partition shown in Fig. 3. Each circle represents one intermediate node. The value within the circle is the index of the node and the inequality corresponding to this intermediate node appears above the node. If the inequality is true, the right sub-tree is chosen to continue for online tree search; otherwise, the search goes to left-subtree. Each leaf node (rectangle in Fig. 4) corresponds to one linear function and a polyhedron in the space of varying parameters, which might include parts of multiple regions. For any given value of varying parameters belonging to this polyhedron, the corresponding linear function will be evaluated to obtain the optimal rate configuration. One leaf node might correspond to multiple regions. In such case, all those regions must have the same optimal solution. Consequently, only one region number needs be stored in the leaf node. For instance, the left child of node 4 is related to both region 9 and region 10, and region number 9 is used for online search. The depth of the tree seen in Fig. 4 is equal to 4. With the help of binary tree, we only need to evaluate at most 4 linear inequalities to locate the region and obtain the optimal solution for online rate adaptation.

4.3.2 Online components

Online rate adaptation is triggered by dynamic arrival and departure of *unadaptable* tasks, such as mission critical tasks with fixed rates. Online rate adaptation works as following:

1. **Trigger**: The Trigger calculates D based on the execution times and rates of the newly arrived tasks or departed tasks and sends the new value of D to the Search Routine.
2. **Search Routine**: After receiving D from the Trigger, the Search Routine traverses the binary tree to locate the region that the current value of D_N belongs to, and then passes the region number to the Evaluator.
3. **Evaluator**: The Evaluator computes the new value of X by evaluating the linear function of the region located by the Search Routine. It then sends the new value of X to Actuators.
4. **Actuator**: the Actuators change the task rates based on the new value of X. If the new task rate of T_i is zero, T_i will be evicted.

Example For the example workload defined in Fig. 1 and Table 1, suppose all 3 tasks are running and the current value of D_N is $[0.7\ 0.5]^T$. The Search Routine goes through the binary tree shown in 4 and ends with region 2. The evaluator then obtains the optimal integer solution $X = [1\ 1\ 0]^T$ corresponding to region 2 and passes it to the actuators on P_1 and P_2. T_1 and T_2 will continue running, but T_3 will be evicted by the actuator on P_2.

4.4 Complexity analysis

In this section we analyze the complexity of the MPRA algorithm. The complexity of the offline part is exponential in the number of decision variables (Murty 1980), which is equal to $\bar{m}$ for discrete rate adaptation, where $\bar{m} = \sum_{1\le i\le m} k_i$, m is the number of the adaptable tasks, and k_i is the number of none-zero rates of task T_i. Note that the exponential complexity is unavoidable in order to get optimal solutions due to the fact that the discrete rate adaptation is an NP-hard problem. A key advantage of MPRA is that it only incurs exponential complexity in the *offline* part which is not time critical and can use significant computing resources. In the following, our analysis focuses on the online search routine and the evaluation of the explicit solution, which dominate the online complexity of MPRA.

The complexity of the online search routine depends on N_r, the number of non-overlapping regions generated by the mp-MILP Solver. We first analyze the mp-MILP algorithm to calculate N_r. The mp-MILP Solver implements the Branch and Bound algorithm presented in Acevedo and Pistikopoulos (1999). There will be $2^{\bar{m}}$ leaf nodes in the enumeration tree. For each leaf node, all $\bar{m}$ binary variables have been fixed and the problem is relaxed to an mpLP problem. Based on the results in Bemporad et al. (2002), the upper bound to the number of regions for one leaf node is $n_r \le n + 1$, where n is the number of processors.

The optimal PWA function of the mp-MILP problem is obtained by removing the overlap among the regions for all leaf nodes. One such region can be divided into at most $2^{\bar{m}}$ non-overlapping regions because it can be associated with at most $2^{\bar{m}}$ solutions. After eliminating the intersection among different regions, we get all N_r non-overlapping regions, which represent a partition of the entire space of D_N. N_r is bounded by

$$N_r \le 2^{\bar{m}} \times n_r \times 2^{\bar{m}} \le (n+1)2^{2\bar{m}} \tag{21}$$

The binary tree generated by the Search Tree Generator reduces the complexity of online region search. For a given D_N we only evaluate one linear inequality at each level, which incurs n multiplications, n additions and 1 comparison. Traversing the tree from the root to the bottom, we will end up with a leaf node that gives us the optimal solution. Then we need $2\bar{m}n$ arithmetic operations for the explicit solution evaluation. According to the result in Tondel et al. (2003), the depth of the binary tree, d, is given by

$$d = \left\lceil \frac{\ln N_r}{\ln 1/\alpha} \right\rceil \le \left\lceil \frac{2\bar{m}\ln 2 + \ln(n+1)}{\ln 1/\alpha} \right\rceil \tag{22}$$

where $0.5 \le \alpha < 1$. The constant α is related to how unbalance the binary tree is. A conservative estimate of α is 2/3 based on the result in Tondel et al. (2003). So the worst-case number of arithmetic operations required for online search and evaluation is $(2n+1)d + 2\bar{m}n$. Let $k = \max\{k_1, \dots, k_m\}$. Then $\bar{m} = \sum_{1 \le i \le m} k_i \le km$. Thus, MPRA has time complexity $O(n\log(n)) + O(mn)$, where m is the number of adaptable tasks and n is the number of the processors.

The memory complexity of MPRA differs between the offline phase and the online phase. In the offline phase the memory cost is exponential to the number of decision variables, i.e., the set of discrete rates of all tasks. In the online phase, the system needs to store all non-overlapping regions and the number of non-overlapping regions is presented in (21). Given its high memory complexity (especially in the offline phase), MPRA effectively trades memory footprint for run-time speed up. As a result it is particularly suitable for systems where reconfiguration speed and quality is more important than memory footprint.

5 Simulation

We have evaluated MPRA through both simulations and experiments on a distributed testbed. We present the simulation results in this section. The empirical results will be presented in the next section.

5.1 Simulation setup

In this section, we present simulation results for both admission control and discrete rate adaptation. Our simulation environment is composed of an event-driven simulator implemented in C++ and the online part of MPRA. The offline pre-processing of MPRA is done in MATLAB.

In our simulation, the subtasks on each processor are scheduled by the Rate Monotonic scheduling (RMS) algorithm (Liu and Layland 1973). Each task's end-to-end deadline is m_i/r_i, where m_i is the number of subtasks of task T_i and r_i is the current rate of the task. The deadline of each task is evenly divided into subdeadlines for its subtasks. The resulting subdeadline of each subtask T_{ij} equals its period, $1/r_i$. Hence we choose the schedulable utilization bound of RMS (Liu and Layland 1973) as the utilization bound on each processor: $b_i = n_i(2^{1/n_i} - 1)$, $1 \le i \le n$, where n_i is the number of subtasks on P_i. MPRA can also be used with other scheduling policies and their suitable utilization bounds.

We develop a workload generator to create end-to-end tasks and the workload for each set of the experiments. In our simulation, each adaptable task has three rate options. We assume all adaptable tasks can be evicted, i.e., $r_i^{(0)} = 0$, $1 \le i \le m$. $r_i^{(1)}$ of task T_i is the reciprocal of task period τ_i, which follows a uniform distribution between 100 ms and 1100 ms. Each task has two non-zero rate options in the experiments of discrete rate adaptation, where the ratio $r_i^{(2)}/r_i^{(1)}$ is uniformly distributed between 1.5 and 3. The task utility value $q_i^{(0)}$ of T_i when the task is evicted is zero and $q_i^{(1)}$ at rate $r_i^{(1)}$ is randomly generated using a uniform distribution between 0.5 and 2. The ratio of the utilities at different rates, $q_i^{(2)}/q_i^{(1)}$ is uniformly distributed between 1.5 and 3. All weights are set to 1 for simplicity in our simulation, i.e.,

$w_i = 1, 1 \leq i \leq m$. The number of subtasks of each task ranges from 1 to 4 and all subtasks are randomly allocated on all processors. The worst-case execution time c_{ij} of subtask T_{ij} is obtained by $c_{ij} = u_{ij}\tau_i$, where u_{ij}, the utilization of T_{ij}, is uniformly distributed from 0.05 to 0.2.

5.2 Baselines

We compare MPRA against three existing algorithms: **bintprog**, **amrmd1** (Lee et al. 1999b), and **amrmd_dp** (Ghosh et al. 2003). **bintprog** is a binary integer linear programming solver provided by the commercial Optimization Toolbox of MATLAB 7. **bintprog** is a representative optimization solver that can produce optimal solutions, which is used to validate the optimality of MPRA. **amrmd1** and **amrmd_dp**, where **amrmd** stands for Approximate Multi-Resource Multi-Dimensional Algorithm, are two representative efficient heuristic algorithms for utility optimization in real-time systems. **amrmd_dp** can perform better than **amrmd1** in terms of utility at the cost of longer execution time than **amrmd1**. The authors also present another algorithm called **amrmd_cm** to address the co-located point problem of **amrmd1** in Ghosh et al. (2003). It performs exactly the same as **amrmd1** here because no co-located points exist in the discrete rate adaptation problem. Note that **amrmd1** and **amrmd_dp** may produce suboptimal solutions and do not have theoretical error bounds as mentioned in Lee and Siewiorek (1998).

5.3 Performance metrics

In our experiments, online adaptation operations are triggered by arrivals of unadaptable tasks. The performance metric used throughout the simulation is *utility improvement*, δ, which is defined by $\delta = (Q_{MPRA} - Q_b)/Q_b$, where Q_{MPRA} and Q_b are the system utilities produced by MPRA and a baseline algorithm, respectively, after they perform the online adaptation in response to the same new task arrivals.

In order to evaluate the efficiency of MPRA, we also investigate its online execution time and compare it with three baselines. The execution times are measured on a 2.52 GHz Pentium 4 PC with 1 GB RAM. To achieve fine grained measurements, we use the high resolution timer *gethrtime* provided by ACE (Center for Distributed Object Computing 2012). This function uses an OS-specific high-resolution timer that returns the number of clock cycles since the CPU is powered up or reset. The *gethrtime* function has a low overhead and is based on a 64 bit clock cycle counter on Pentium processors. To estimate the average computation overhead of an online adaptation operation, we run each online execution for 100 times as a subroutine. The result is then divided by 100 to get the execution time of a single execution.

5.4 End-to-end admission control

We randomly generated 20 workloads in the simulation of end-to-end admission control. Each workload comprises 8 end-to-end adaptable tasks executing on 4 processors. In the following, we present a set of experiments to evaluate the performance of the four algorithms in the presence of arrivals of unadaptable tasks, which are mission critical periodic tasks that must be executed at the cost of other tasks.

We run a set of experiments by varying the CPU utilization of the new arrival task from 0.2 to 0.5. Four identical new tasks are activated after 250000 time units on four processors simultaneously. Consequently, online admission control is triggered to maximize system utility while enforcing the utilization bounds. We repeated the same set of experiments 20 times with each run for one workload. We observe that MPRA and **bintprog** always produce the same optimal rates and hence achieve the same system utility in all the experiments. These results are consistent with the optimality of MPRA. We plot the average and maximum *utility improvements* achieved by MPRA over **amrmd1** and **amrmd_dp** under different utilization variations caused by the new tasks in Fig. 5. Each data point and the corresponding confidence interval in Fig. 5 is calculated from all 20 *utility improvement* results obtained in 20 runs under a given workload variation. As shown in Figs. 5(a, b) MPRA consistently achieves higher system utility than both **amrmd1** and **amrmd_dp** under different degrees of workload variations. Moreover, as seen in Fig. 5(c) MPRA can improve the system utility by as high as 26% and 19% over **amrmd1** and **amrmd_dp**, respectively. Our results demonstrate that, while state-of-the-art heuristics such as **amrmd1** and **amrmd_dp** may achieve good (but suboptimal) performance on average, they may result in significantly lower system utility in certain cases. This observation is consistent with the fact that the heuristics do not have analytical bounds on the distance from optimal solutions. In contrast, a fundamental benefit of MPRA is that it can always achieve *optimal* system utility in face of workload variations. The analytical guarantee on optimal system utilities can be highly desirable to dynamic mission-critical applications.

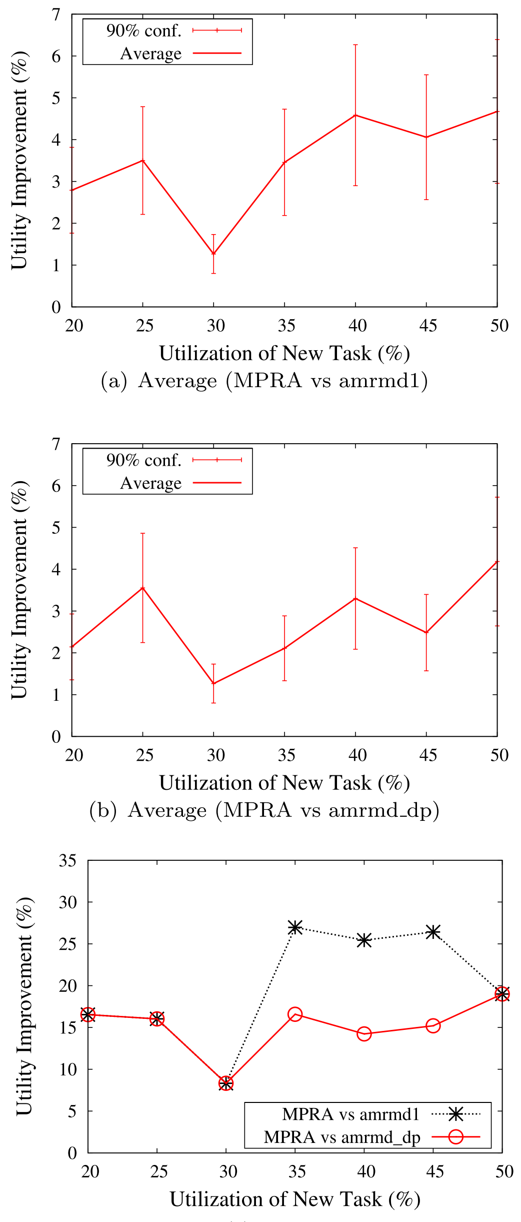
Figure 6 plots the average online execution times of all four algorithms. MPRA, **amrmd1**, and **amrmd_dp** are more than two orders of magnitude faster than **bintprog**. For instance, when the new task has a utilization of 30 %, MPRA incurs an overhead of only 100 *microseconds*, while **bintprog** needs about 100 *milliseconds* to generate the same optimal rate assignments. The results show that MPRA can provide optimal admission control for end-to-end tasks with comparable online overhead as efficient suboptimal heuristics.

5.5 Discrete rate adaptation

In the simulation of discrete rate adaptation each workload includes 6 end-to-end adaptable tasks executing on 4 processors. The results are based on 20 randomly generated workloads.

We use a similar set of experiments as that presented in the previous section to investigate the performance of the four algorithms when applied for discrete rate adaptation. Each adaptable task has three rate choices. We intentionally choose a small set of rate choices for each adaptable task in order to stress-test MPRA's capability to support *discrete* rate options. The fewer rates per task, the more significantly does the problem deviate from the continuous case. In our simulation, all tasks are running at the lower rate at the beginning.

In this set of experiments, to generate workload variations, an unadaptable task arrives at each of the four processors simultaneously at 250000 time unit. When new tasks arrive, rate adaptation is triggered to enforce the desired utilization bound and



(a) Average (MPRA vs amrmd1)

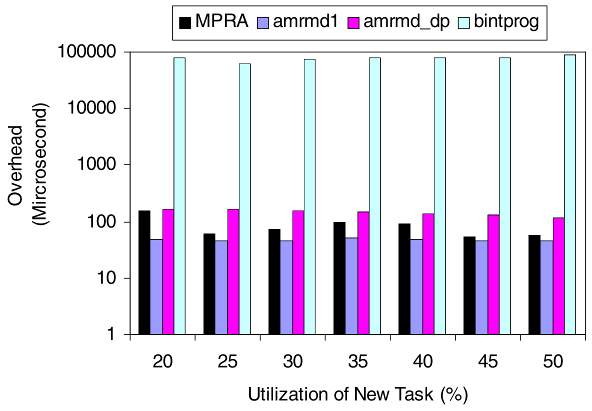
(b) Average (MPRA vs amrmd_dp)

(c) Max

Fig. 5 Admission control: utility improvement over heuristics

maximize system utility. Figure 7 plots the *utility improvements* achieved by MPRA over **amrmd1** and **amrmd_dp** as the utilization of the new task increases from 0.2 to 0.5 in different runs. Similar to results for admission control, MPRA consistently achieves same utilities as **bintprog** and outperforms both **amrmd1** and **amrmd_dp**

Fig. 6 Admission control: online overhead



in terms of system utility. MPRA achieves as high as 35 % utility improvement over both **amrmd1** and **amrmd_dp**.

The average execution-times of the four approaches when applied for discrete rate adaptation are shown in Fig. 8. MPRA's online overhead is more than two orders of magnitude lower than that of **bintprog** while generating the same optimal solutions. MPRA remains comparable to **amrmd1** and **amrmd_dp** in terms of online overhead. As shown in Fig. 8, the overhead introduced by MPRA is about 100 *microseconds*, which is negligible when compared to both (i) the time scale of deadlines and periods of end-to-end tasks in many distributed real-time systems and (ii) the overhead of more than 20 *milliseconds* incurred by adjusting task rates on middleware systems (Wang et al. 2007).

6 Empirical evaluation

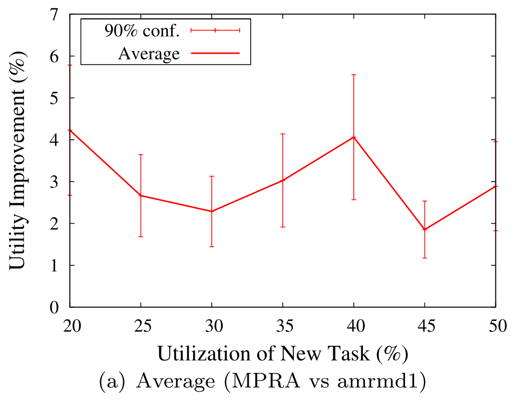
In this section, we introduce the implementation of MPRA in a real-time middleware system and present the empirical results.

6.1 Middleware implementation

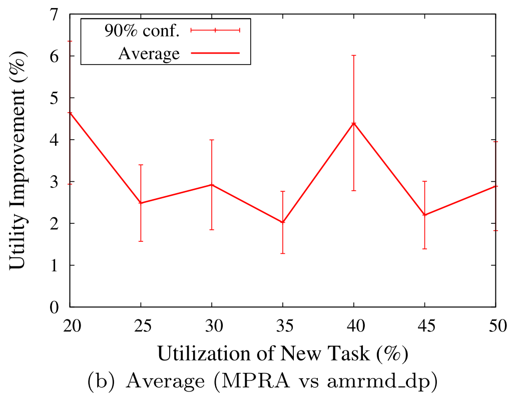
MPRA has been integrated into the FC-ORB middleware (Wang et al. 2005b). The middleware architecture with the extension of MPRA is shown in Fig. 9.

We first give a brief overview of FC-ORB middleware. FC-ORB implements an end-to-end real-time Object Request Broker (ORB) service that supports end-to-end real-time tasks. Each subtask is executed by a separate thread. The first subtask of a task is associated 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 through a TCP connection to the succeeding subtask that might be located on another processor. FC-ORB implements the release guard protocol with a FIFO waiting queue and one-shot ACE timers. Upon receiving a remote operation request, a subtask compares the current time with the last invocation time of this operation.

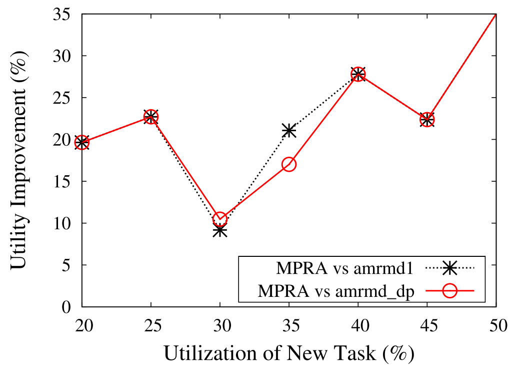
Fig. 7 Rate adaptation: utility improvement over heuristics



(a) Average (MPRA vs amrmd1)



(b) Average (MPRA vs amrmd_dp)



(c) Max

Based on the release guard rules (Sun and Liu 1996), the subtask either immediately invokes the requested operation or enqueues this request to the waiting queue if the request arrives too early. When the request is enqueued, a one-shot ACE timer is registered with the reactor to trigger the requested operation at the time that equals the last invocation time plus the task's period. After the one-shot timer fires and the

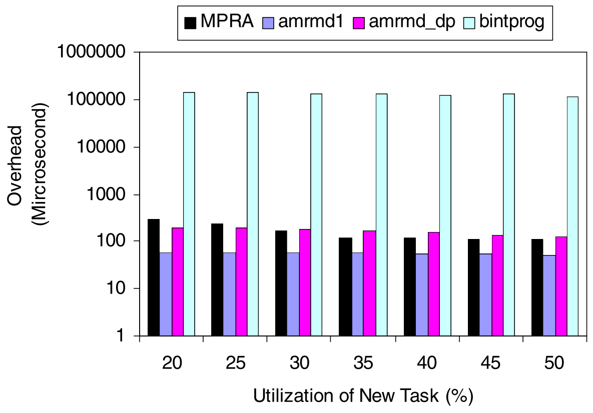


Fig. 8 Rate adaptation: online overhead

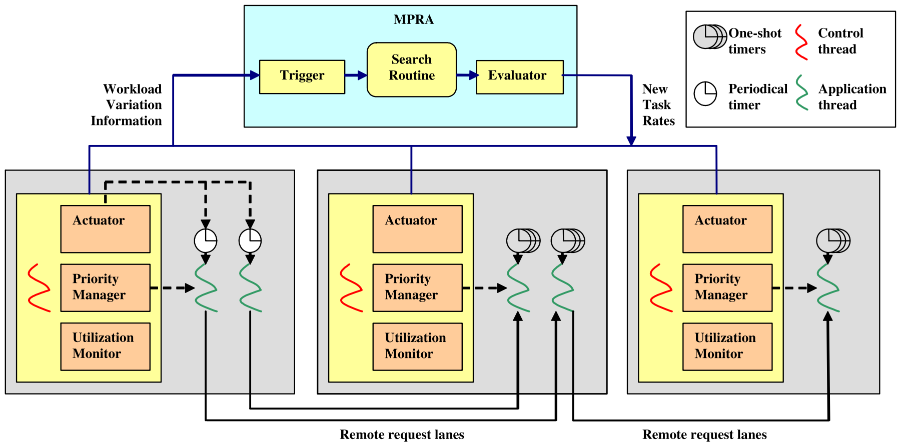


Fig. 9 Middleware architecture

enqueued request is served, a remote operation request is sent to the next subtask in the end-to-end task chain. FC-ORB employs a priority manager on each processor to assign priorities to local subtasks based on a real-time scheduling algorithm (e.g., RMS). FC-ORB also provides a rate modulator and a utilization monitor on each processor. The rate modulator receives the new rates for the tasks with their first subtasks located on the local processor, and resets the timer interval of the first subtask of each task whose invocation rate has been changed. The utilization monitor uses the /proc/stat file in Linux to estimate the CPU utilization during a specified time interval.

As shown in Fig. 9, MPRA is implemented as an independent process that can be deployed on a separate processor or on an application processor. Application processors run FC-ORB to execute the given real-time workloads. Every application processor in the system connects with MPRA through a TCP connection when the system starts. We extend the rate modulator component provided by FC-ORB to support admission control. A task is effectively rejected from the system when the timer associated with its first subtask is stopped. The extended component is called *actuator* in Fig. 9.

The MPRA algorithm is triggered by dynamic arrival and departure of unadaptable tasks. When such event happens, the system works as follows: (1) the actuator on the processor hosting the first subtask of each arriving/departing task sends the execution times and rates of the task to the trigger component of MPRA through the TCP connections; (2) MPRA computes the new task rates and sends the new rates to the actuators on all processors hosting adaptable tasks; (3) the actuators on processors that host the first subtasks of adaptable tasks change the rates of the first subtasks or cancel the periodic timers of the first subtasks if the corresponding end-to-end tasks need to be evicted.

6.2 Empirical results

In this section, we present the results of two sets of experiments. To make the experimental results comparable with the simulations, we employed the same method to generate the workload as one used for the simulations (see Sect. 5.1). All experiments are conducted on a distributed testbed with five machines. All applications and the ORB service run on a Linux cluster composed of four Pentium-IV machines: Ron, Harry, Norbert, and Hermione. Ron and Hermione are 2.80 GHz, and Harry and Norbert are 2.53 GHz. All four machines are equipped with 512 KB cache and 512 MB RAM, and run KURT Linux 2.4.22. MPRA is located on another Pentium-IV 2.53 GHz machine with 512 KB cache and 1 G RAM. The MPRA machine runs Windows XP Professional. The four machines in the cluster are connected via an internal switch and communicate with the MPRA machine through the departmental 100 Mbps LAN.

Since amrmd1 consistently achieved better performance and at a smaller overhead than amrmd_dp (Figs. 5–8), we compare MPRA with only amrmd1 in experiments. We also excluded binprog as a baseline because it always delivered the same utility as MPRA as discussed in the simulation section.

In the first set of experiments, we compare the performance of MPRA and **amrmd1** when applied for end-to-end admission control. An *unadaptable* task with utilization of 0.2 is activated on each processor at 250, 500, 750, and 1000 second, respectively, which triggers online admission control four times. Figure 10 plots CPU utilizations and the system utility during a run. The CPU utilizations are collected by the utilization monitors at a sampling period of 10 seconds. The system utility is the sum of the utilities of all *adaptable* tasks only. As a result, the system utility drops when it reduces the rates or evicts those tasks to accommodate the arrival of new *unadaptable* tasks. As seen in Figs. 10(a, b) MPRA and **amrmd1** enforce the utilization bounds on all four processors by evicting tasks in response to the workload increase. Figure 10(c) shows that MPRA achieves higher system utility than **amrmd1**. For example, after the new task arrives on Norbert at 750 second, the utility achieved by MPRA remains 8.02 while the utility achieved by **amrmd1** drops from 7.63 to 6.88.

In the second set of experiments, MPRA and **amrmd1** are applied for discrete rate adaptation. All tasks are initialized to run at their lowest rates during [0 s, 250 s]. An *unadaptable* task arrives at each processor and triggers rate adaptation at 250, 500,

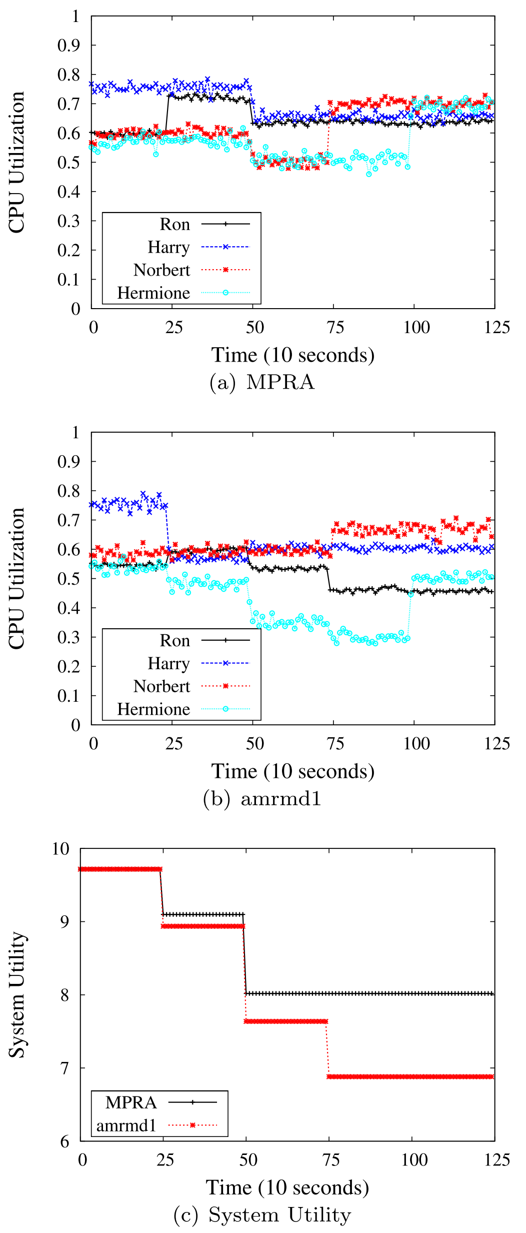
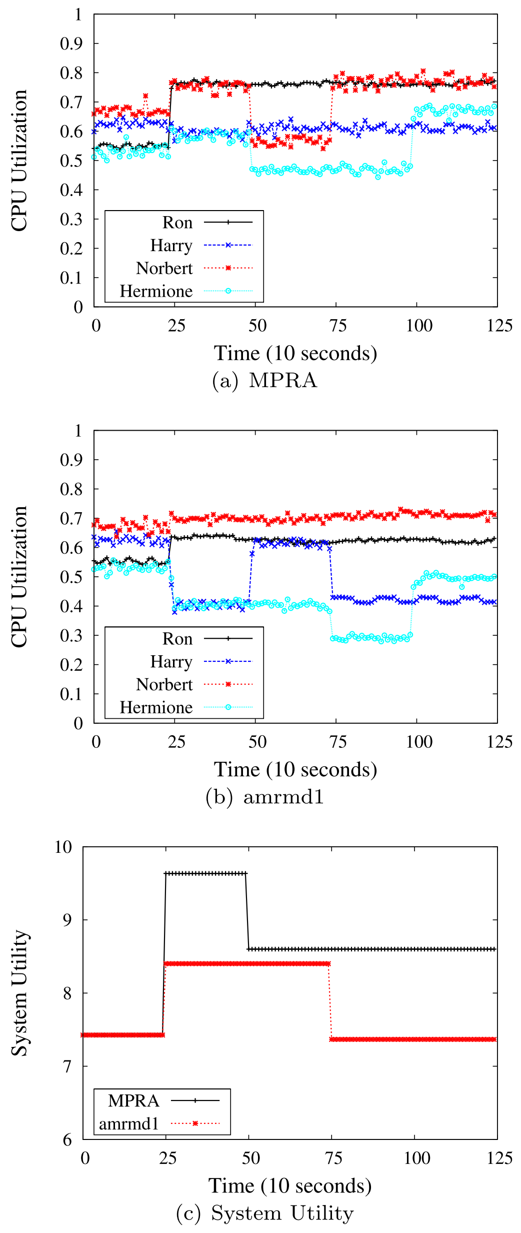


Fig. 10 Admission control: system performance under separate task arrivals

750, and 1000 second, respectively. The CPU utilization of each *unadaptable* task is 0.2. As shown in Fig. 11, both algorithms enforce the utilization bounds on all processors in face of new task arrivals. However, MPRA generates optimal rates that result in higher system utilities than **amrmd1** in response to new task arrivals.

Fig. 11 Rate adaptation: system performance under separate task arrivals



7 Conclusions

We have developed the MPRA algorithm for optimal and efficient discrete rate adaptation in distributed real-time systems. We first transform the discrete rate adaptation problem to an mp-MILP problem. We then present the design and complexity analysis which prove that MPRA can have *polynomial* online complexity through offline preprocessing. Simulation results demonstrate that MPRA maximizes the system utility in face of workload variations, with the online execution time more than two orders of magnitude lower than a representative optimization solver. Moreover, it consistently outperforms efficient heuristics in terms of system utility at comparable online overhead. The advantages of MPRA have also been demonstrated through its implementation on the FC-ORB middleware and experiments on a distributed testbed.

While we focus on admission control and discrete rate adaptation in this paper, the multiparametric approach may be applicable to a broad range of adaptive systems with discrete configurations such as task reallocation or dynamic voltage scaling. MPRA can potentially be integrated with feedback utilization control approaches specifically designed to handle varying task execution times. The multi-parametric programming framework can be employed to implement efficient online solutions to Model Predictive Control (MPC), the control approach adopted by existing feedback utilization algorithms such as EUCON (Lu et al. 2005).

References

Abdelzaher TF, Atkins EM, Shin KG (2000) QoS negotiation in real-time systems and its application to automated flight control. IEEE Trans Comput 49(11):1170–1183

Abdelzaher TF, Shin KG, Bhatti N (2002) Performance guarantees for web server end-systems: a control-theoretical approach. IEEE Trans Parallel Distrib Syst 13(1):80–96

Abdelzaher T, Stankovic J, Lu C, Zhang R, Lu Y (2003) Feedback performance control in software services. IEEE Control Syst 23(3):74–90

Acevedo J, Pistikopoulos E (1999) An algorithm for multiparametric mixed-integer linear programming problems. Oper Res Lett 24(10):139–148

Bemporad A, Borrelli F, Morari M (2002) Model predictive control based on linear programming—the explicit solution. IEEE Trans Autom Control 47(12):1974–1985

Brandt SA, Nutt GJ (2002) Flexible soft real-time processing in middleware. Real-Time Syst 22(1–2):77–118

Brandt S, Nutt G, Berk T, Mankovich J (1998) A dynamic quality of service middleware agent for mediating application resource usage. In: IEEE real-time systems symposium, pp 307–316

Buttazzo GC, Lipari G, Caccamo M, Abeni L (2002) Elastic scheduling for flexible workload management. IEEE Trans Comput 51(3):289–302

Center for Distributed Object Computing (Washington University) (2012) The ADAPTIVE Communication Environment (ACE). www.cs.wustl.edu/~schmidt/ACE.html

Cervin A, Eker J, Bernhardsson B, Årzen KE (2002) Feedback-feedforward scheduling of control tasks. Real-Time Syst 23(1–2):25–53

Chen Y, Lu C, Koutsoukos X (2007) Optimal discrete rate adaptation for distributed real-time systems. In: IEEE real-time systems symposium, pp 181–192

Chen J, Tan R, Xing G, Wang X, Fu X (2010) Fidelity-aware utilization control for cyber-physical surveillance systems. In: IEEE real-time systems symposium, pp 117–126

Fu Y, Kottenstette N, Chen Y, Lu C, Koutsoukos XD, Wangh H (2010a) Feedback Thermal control for real-time systems. In: RTAS, pp 111–120

Fu Y, Lu C, Wang H (2010b) Control-theoretic thermal balancing for clusters. In: IPDPS, pp 1–11

Gal T, Nedoma J (1972) Multiparametric linear programming. Manag Sci 18:406–442

Ghosh S, Rajkumar RR, Hansen J, Lehoczky J (2003) Scalable resource allocation for multi-processor QoS optimization. In: International conference on distributed computing systems, pp 174–183

Ibarra OH, Kim CE (1975) Fast approximation algorithms for the knapsack and sum of subset problems. J ACM 22(4):463–468

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

Koutsoukos X, Tekumalla R, Natarajan B, Lu C (2005) Hybrid supervisory utilization control of real-time systems. In: IEEE real-time and embedded technology and applications symposium, pp 12–21

Kvasnica M, Grieder P, Baotić M (2004) Multi-Parametric Toolbox (MPT)
Lee C, Siewiorek D (1998) An Approach for Quality of Service Management. Technical Report CMU-CS-98-165, Computer Science Department, CMU
Lee C, Lehoczky J, Rajkumar R, Siewiorek DP (1999a) On quality of service optimization with discrete QoS options. In: IEEE real time technology and applications symposium, pp 276–286
Lee C, Lehoczky JP, Siewiorek DP, Rajkumar R, Hansen JP (1999b) A scalable solution to the multi-resource QoS problem. In: IEEE real-time systems symposium, pp 315–326
Lee CG, Shih CS, Sha L (2004) Online QoS optimization using service classes in surveillance radar systems. Real-Time Syst 28(1):5–37
Lehoczky JP (1990) Fixed priority scheduling of periodic task sets with arbitrary deadlines. In: IEEE real-time systems symposium, pp 201–213
Lindberg M, Årzén KE (2010) Feedback control of cyber-physical systems with multi resource dependencies and model uncertainties. In: RTSS, pp 85–94
Liu C, Layland J (1973) Scheduling algorithms for multiprogramming in a hard-real-time environment. J ACM 20(1):46–61
Lu C, Stankovic J, Tao G, Son S (2002) Feedback control real-time scheduling: framework, modeling, and algorithms. Real-Time Syst 23(1/2):85–126
Lu C, Wang X, Gill C (2003) Feedback Control Real-Time Scheduling in ORB Middleware. In: IEEE Real-Time and Embedded Technology and Applications Symposium, pp 37–48
Lu C, Wang X, Koutsoukos X (2005) Feedback utilization control in distributed real-time systems with end-to-end tasks. IEEE Trans Parallel Distrib Syst 16(6):550–561
Martello S, Toth P (1990) Knapsack problems: algorithms and computer implementations. Wiley, New York
Murty KG (1980) Computational complexity of parametric linear programming. Math Program 19(1):213–219
Natale MD, Stankovic J (1994) Dynamic end-to-end guarantees in distributed real-time systems. In: IEEE real-time systems symposium, pp 216–227
Rajkumar R, Lee C, Lehoczky J, Siewiorek D (1997) A resource allocation model for QoS management. In: IEEE real-time systems symposium, pp 298–307
Rajkumar R, Lee C, Lehoczky JP, Siewiorek DP (1998) Practical solutions for QoS-based resource allocation. In: IEEE real-time systems symposium, pp 296–306
Sahni S (1975) Approximate algorithms for the 0/1 knapsack problem. J ACM 22(1):115–124
Steere DC, Goel A, Gruenberg J, McNamee D, Pu C, Walpole J (1999) A feedback-driven proportion allocator for real-rate scheduling. In: Operating systems design and implementation, pp 145–158
Sun J, Liu JWS (1996) Synchronization protocols in distributed real-time systems. In: International conference on distributed computing systems, pp 38–45
Tokuda H, Kitayama T (1994) Dynamic QoS control based on real-time threads. In: NOSSDAV, vol 93. Springer, London, pp 114–123
Tondel P, Johansen TA, Bemporad A (2003) Evaluation of piecewise affine control via binary search tree. Automatica 39(5):945–950
Wang X, Jia D, Lu C, Koutsoukos X (2005a) Decentralized utilization control in distributed real-time systems. In: IEEE real-time systems symposium, pp 133–142
Wang X, Lu C, Koutsoukos X (2005b) Enhancing the robustness of distributed real-time middleware via end-to-end utilization control. In: IEEE real-time systems symposium, pp 189–199
Wang X, Chen Y, Lu C, Koutsoukos X (2007) FC-ORB: a robust distributed real-time embedded middleware with end-to-end utilization control. J Syst Softw 80(7):938–950
Wang X, Fu X, Liu X, Gu Z (2009) Power-aware CPU utilization control for distributed real-time systems. In: IEEE real-time and embedded technology and applications symposium, pp 233–242

Yingming Chen received the B.S. 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.

Chenyang Lu is a Professor of Computer Science and Engineering at Washington University in St. Louis. He is Editor-in-Chief of ACM Transactions on Sensor Networks and Associate Editor of Real-Time Systems. He also served as Program Chair of IEEE Real-Time Systems Symposium (RTSS 2012) and ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS 2012). He is the author and co-author of over 100 papers. He received the Ph.D. from University of Virginia in 2001, the M.S. degree

from Chinese Academy of Sciences in 1997, and the B.S. degree from University of Science and Technology of China in 1995, all in computer science. His research interests include real-time systems, wireless sensor networks and cyber-physical systems.

Xenofon D. Koutsoukos is an Associate Professor in the Department of Electrical Engineering and Computer Science at Vanderbilt University. He received the diploma in electrical and computer engineering from the National Technical University of Athens in 1993, the M.S. degrees in electrical engineering and applied mathematics, and the Ph.D. degree in electrical engineering from the University of Notre Dame in 1998 and 2000, respectively. He has published numerous journal and conference papers and is co-inventor of four US patents. His research work is in the area of cyber-physical systems with emphasis on formal methods, distributed algorithms, diagnosis and fault tolerance, and adaptive resource management.