Report Number: WUCSE-2007-35

2007

Optimal Discrete Rate Adaptation for Distributed Real-Time Systems

Yingming Chen, Chenyang Lu, and Xenofon Kutsoukos

Many distributed real-time systems face the challenge of dynamically maximizing system utility in response to fluctuations in system workload. 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 such as dynamic task arrivals. 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... **Read complete abstract on page 2.**

Optimal Discrete Rate Adaptation for Distributed Real-Time Systems

Yingming Chen, Chenyang Lu, and Xenofon Kutsoukos

Complete Abstract:

Many distributed real-time systems face the challenge of dynamically maximizing system utility in response to fluctuations in system workload. 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 such as dynamic task arrivals. 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.

Department of Computer Science & Engineering

2007-35

Optimal Discrete Rate Adaptation for Distributed Real-Time Systems

Authors: Yingming Chen, Chenyang Lu, and Xenofon Koutsoukos

Corresponding Author: yingming@cse.wustl.edu

Abstract: Many distributed real-time systems face the challenge of dynamically maximizing system utility in response to fluctuations in system workload. 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 such as dynamic task arrivals. 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.

Type of Report: Other

Optimal Discrete Rate Adaptation for Distributed Real-Time Systems

Yingming Chen Chenyang Lu
Department of Computer Science and Engineering
Washington University in St. Louis
St. Louis, MO 63130
{*yingming, lu*}*@cse.wustl.edu*

Xenofon Koutsoukos
Department of Electrical Engineering
and Computer Science
Vanderbilt University
Nashville, TN 37235
xenofon.koutsoukos@vanderbilt.edu

Abstract

Many distributed real-time systems face the challenge of dynamically maximizing system utility in response to fluctuations in system workload. 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 such as dynamic task arrivals. 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.

I. Introduction

An increasing number of distributed real-time systems operate in dynamic environments where system workload may change at run time [1]. 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. In particular, such systems need to enforce desired CPU utilization bounds on multiple processors in order to provide overload protection and meet end-to-end deadlines. Therefore, online adaptation must be adopted to handle workload changes in such systems.

Online adaptation introduces several important challenges. 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. Second, many common adaptation strategies only support *discrete* options. For example, an admission controller must make binary decision (admission/rejection) on a task. While task rate adaptation can allow a system to adapt at a finer granularity [2][3][4][5][6][7], many real-time applications (e.g., avionics [8] 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 [9]. Furthermore, despite the difficulty of such problems, a real-time system must adapt to 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 [9][10]. 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 [11][9][8][12]. However, these algorithms can only produce sub-optimal 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 task rate adaptation as the online adaptation mechanism, which is supported by a broad range of real-time applications, such as digital control [3], video streaming, and avionics [8]. Specifically, MPRA is designed to handle end-to-end tasks that may only execute at a discrete set of rates on multiple processors. This task model introduces significant challenges to optimal online adaptation algorithms.

The key novelty and advantage of our approach is that it can *efficiently* produce *optimal* solutions online in face of workload changes caused by dynamic task arrivals and departures. The MPRA algorithm is based on multiparametric mixed-integer linear programming (mp-MILP) [13]. 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 three-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.

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 complexity analysis of our algorithm. Section 5 provides simulation results. Finally, Section 6 concludes this paper.

II. 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) [14] 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 [9][10]. 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 sub-optimal solutions [11][9][10][15][16]. Specifically, they presented heuristic algorithms with bounded approximation ratio for the single-resource case [10][15]. However, the heuristic algorithms for multi-resource problems do not have analytical bounds on the approximation ratio [15]. Note that the multi-resource case is common in distributed real-time systems in which each processor is a separate resource.

Various system-wide schemes have been studied to improve system utility. The authors in [17][18][19] 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 [8]. All of the projects developed middleware systems that aim to improve system utility by dynamically adjusting the QoS levels of applications. However, they 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 distributed real-time systems [12]. 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 [3][2][7] and distributed real-time systems [5][20]. All the above solutions assume that task rates can be adjusted in a *continuous* range. As discussed in Section 1, this assumption does not hold in many applications that only support *discrete* configurations. HySUCON [6] 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 [6]) is designed to handle discrete rates. Moreover, none of them can maximize system utility.

III. Problem Formulation

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

A. End-to-End Task Model

The system is comprised of m periodic tasks $\{T_i|1 \leq i \leq m\}$ executing on n processors $\{P_i|1 \leq i \leq n\}$. Task T_i is composed of a graph of subtasks $\{T_{ij}|1 \leq j \leq m_i\}$ that may be located on different processors. We denote the set of subtasks of 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 d_i related to its period τ_i. Each subtask T_{ij} has an execution time c_{ij}.

We assume each 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 $R_i = \{r_i^{(0)}, ..., r_i^{(k_i)}\}$ in increasing order. The set of utility options for task T_i is denoted by $Q_i = \{q_i^{(0)}, ..., q_i^{(k_i)}\}$ where $q_i^{(j)}$ is the utility value contributed by T_i when it is configured with $r_i^{(j)}$. Note that we do not make any assumption about a task's utility values. For example, they 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 have two rate choices: zero when the task is evicted and a fixed non-zero rate when task is admitted.

B. Discrete Rate Adaption Problem

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

- R: $R = [r_1, ..., r_m]$ is the task rate vector where r_i is the current invocation rate of task T_i. Therefore we have $r_i \in R_i, 1 \leq i \leq m$.
- Q_s: Q_s is the system utility, i.e., the combined utility of all the 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 $0 \leq w_i \leq 1$, $1 \leq i \leq m$, are weights describing the relative importance of the tasks.
- D: $D = [d_1, ..., d_n]$ is the workload variation vector where d_i is the change to the utilization of the i^{th} processor caused by workload variations, e.g., dynamic arrivals and departures of tasks with fixed rates. The worst-case execution times and rates of the tasks that introduce workload variations are known and can be used to calculate D. For example, denote the set of tasks that newly arrive as S_a and the set of tasks that just 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: $U = [u_1, ..., u_n]$ is the CPU utilization vector where u_i represents the utilization of the i^{th} processor. u_i is calculated by $u_i = d_i + \sum_{1 \leq j \leq m} \sum_{T_{jl} \in S_{ij}} c_{jl} r_j$.
- B: $B = [b_1, ..., b_n]$ is the utilization bound vector where b_i is the utilization bound of the i^{th} 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 via rate adaptation in response to workload changes, i.e.

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

[1] A non-greedy synchronization protocol [21] can be used to remove release jitter of subtasks.

subject to

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

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

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 two real-time requirements:

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 [21], 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 [22][23]. To meet end-to-end deadlines, a user sets the utilization set point of each processor to a value below its schedulable utilization bound. We can apply various subdeadline assignment algorithms [24][25] and schedulable utilization bounds for different task models [22][23] presented in the literature.

Overload protection. Many distributed systems must avoid saturation of processors, which may cause system crash or severe service degradation [26]. On COTS operating systems that support real-time priorities, high utilization by real-time threads may cause kernel starvation [27]. The utilization constraint (3) allows a user to enforce desired utilization bounds for all the processors in a distributed system.

The discrete rate adaptation problem is NP-hard as it can be easily reduced to the 0-1 Knapsack Problem [28]. 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 [29][30]. However, those algorithms can only handle problems for the single-resource case and can not be applied for multi-resource problems.

IV. Design and Analysis of MPRA

In this section, we present the design and analysis of MPRA. We first give a brief overview of the general framework of multiparametric programming. Next, we transform the discrete rate adaptation problem to an mp-MILP problem and design MPRA that instantiates the multiparametric programming approach for optimal and efficient rate adaptation in distributed real-time systems. Finally, we present the complexity analysis of our algorithm.

A. Multiparametric Programming

Multiparametric programming is a general framework for solving mathematical programming problems with constraints that depend on varying parameters [31]. Multiparametric programming includes an offline and an online algorithm. 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) [13]. The algorithm presented in [13] 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

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

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

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

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 $\Xi = \{\theta | G\theta \leq g; \theta \in \Re^s\}$. The optimal solution to this problem is a piecewise affine (PWA) function with a polyhedral partition of the following form

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

where the regions $\Theta_i \stackrel{\triangle}{=} \{\theta \in \Xi : H_i\theta \leq k_i\}$, $i = 1, ..., N_r$ 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 configurations. The key advantage of the multiparametric programming is that, while the offline component 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. To our knowledge MPRA is the first instantiation of the general multiparametric programming approach in the area of real-time systems.

B. Problem Transformation

The key step in the design of MPRA is to transform the discrete rate adaptation problem presented in Section III-B to an mp-MILP problem. We start with the end-to-end admission control problem, which is a special case of discrete rate adaptation, followed by the general case.

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 introduce an admission vector X with m elements to represent rate choices for all 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 $q_i = w_i q_i^{(1)}$, $1 \leq i \leq m$. 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

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

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

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

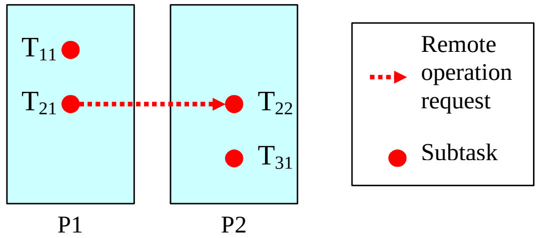


Fig. 1. An Example Workload

Example: Suppose there are two processors and three tasks in the system. As shown in Figure 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. We have

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

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

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 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 task in an appropriate order. For instance, if there are two tasks in the system and each task has two non-zero choices, then $X = [0\ 1\ 1\ 0]$ indicates that task T_1 and T_2 are configured with $r_1^{(2)}$ and $r_2^{(1)}$, respectively. 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 task and contains the information of the non-zero rate options for the task. Again, if there are two tasks in the system and each task has two non-zero choices, then $Z = \left(\begin{smallmatrix} r_1^{(1)} & r_1^{(2)} & 0 & 0 \\ 0 & 0 & r_2^{(1)} & r_2^{(2)} \end{smallmatrix} \right)$.

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

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

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

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

The constraint (18) enforces desired CPU utilization bounds on all processors. The constraint (19) shows that each task only supports a 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 Figure 1 to demonstrate how to formulate a discrete rate adaptation problem. In this example each task has two non-zero rate options. Then $\bar{m} = 6$. We have

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

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

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

The constraint (20) can be described by

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

C. Design of MPRA

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

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 PWA function and then calls the Search Tree Generator to build a binary tree for the representation of the PWA function.

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 fixs 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

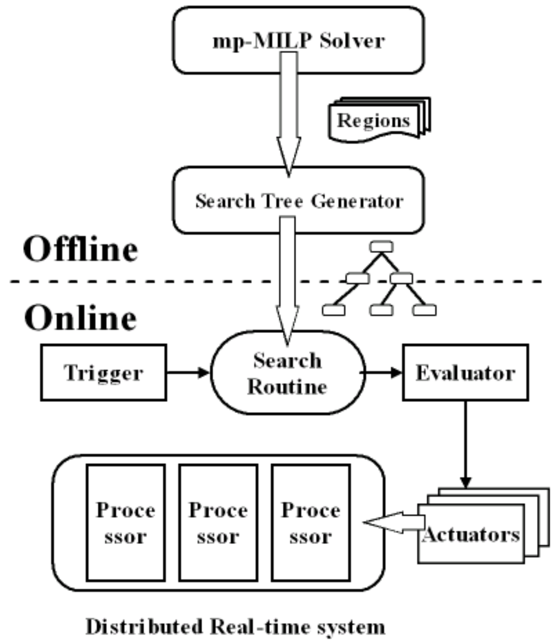


Fig. 2. Overview of MPRA

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.

Search Tree Generator: It generates a binary tree data structure for the representation of 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. An intermediate node contains the affine function 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 maintains 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 select the left or right sub-tree to continue based on the sign. With the help of the binary tree, the time of the evaluation of the PWA function becomes *logarithmic* in the number of regions.

We implemented the offline part of MPRA using the MPT toolbox [32], which provides an mp-MILP solver [13] and a binary tree generator [33].

2) Online Components: Online rate adaptation is triggered by a specified set of events that introduce workload changes. In our implementation rate adaptation is triggered by dynamic arrival and departure of tasks that are not managed by MPRA, 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.

D. 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 [34], 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 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 [13]. 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 [35], 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 [33], the depth of the binary tree, d, is given by

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

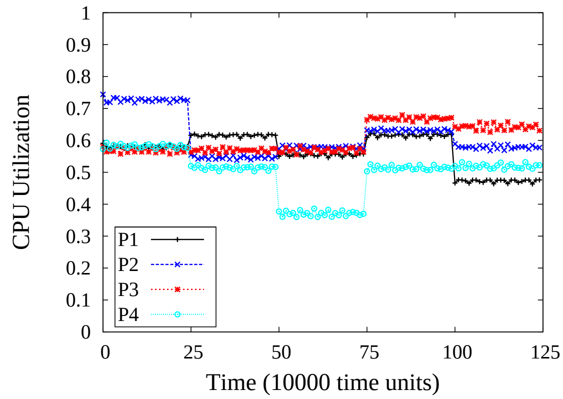
where $0.5 \le \alpha < 1$. The constant α is related to how inbalance the binary tree is. A conservative estimate of α is 2/3 based on the result in [33]. So the worst-case number of arithmetic operations required for online search and evaluation is $(2n+1)d + 2\bar{m}n$. Let $k = \max\{k_1, ..., k_m\}$. Then $\bar{m} = \sum_{1 \le i \le m} k_i \le km$. Thus, MPRA has time complexity $O(nlog(n)) + O(mn)$, where m is the number of tasks and n is the number of the processors.

V. Evaluation

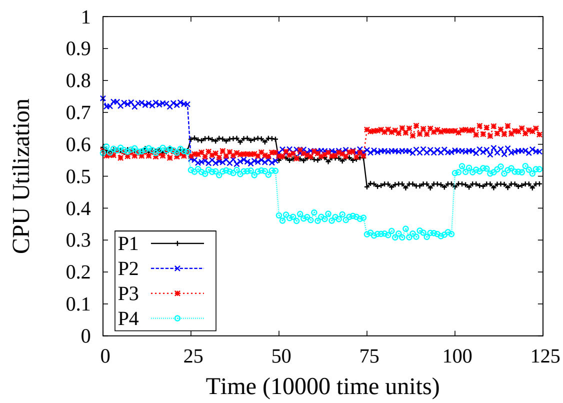
In this section, we present simulation results for both admission control and discrete rate adaptation. Our simulation environment is composed of an event-driven simulator implemented in C++ and the online part of MPRA. The offline pre-processing of MPRA is done in MATLAB.

In our simulation, the subtasks on each processor are scheduled by the Rate Monotonic scheduling (RMS) algorithm [36]. Each task's end-to-end deadline $d_i = m_i/r_i$, 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 resultant subdeadline of each subtask T_{ij} equals to its period, $1/r_i$. Hence we choose the schedulable utilization bound of RMS [36] 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. b_i is dynamically calculated online based on n_i and therefore changes upon task arrivals and departures. MPRA can also be used with other scheduling policies and their suitable utilization bounds.

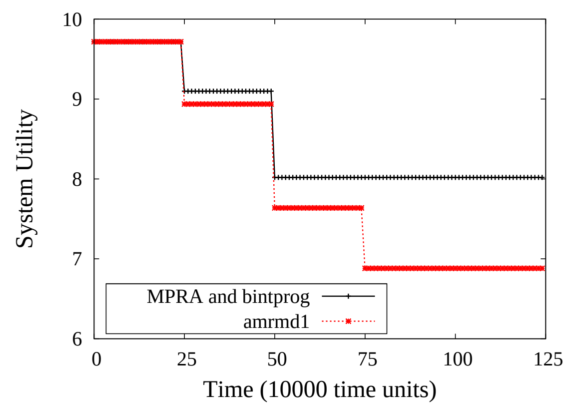
We develop a workload generator to create end-to-end tasks and the workload for each set of the experiments. In our simulation, every task can be evicted, i.e., $r_i^{(0)} = 0$, $1 \le i \le m$. $r_i^{(1)}$ of task T_i is the



(a) MPRA and bintprog



(b) amrmd1

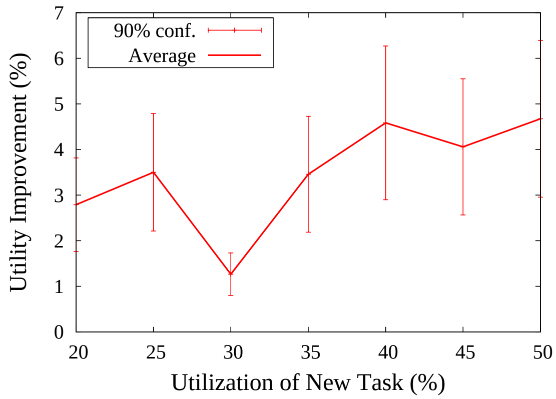


(c) System Utility

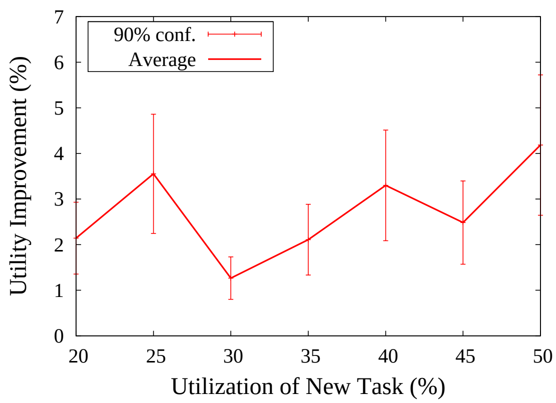
Fig. 3. Admission Control: System Performance under Separate Task Arrivals

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., $w_i = 1, 1 \le i \le m$. 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.

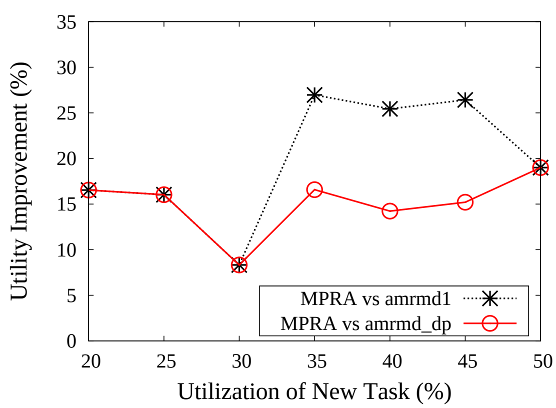
We compare MPRA against three existing algorithms: **bintprog**, **amrmd1** [9], and **amrmd_dp** [16].



(a) Average (MPRA vs amrmd1)



(b) Average (MPRA vs amrmd_dp)



(c) Max

Fig. 4. Admission Control: Utility Improvement over Heuristics

bintprog is a binary integer linear programming solver provided by the commercial Optimization Toolbox from 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**[2]. However, **amrmd1** and **amrmd_dp** may produce sub-optimal solutions and do not have theoretical error bounds as mentioned in [15].

In our experiments, online adaptation operations are triggered by new task arrivals. The performance

[2]The authors also present another algorithm called **amrmd_cm** to address the co-located point problem of **amrmd1** in [16]. It performs exactly the same as **amrmd1** here because no co-located points exist in the discrete rate adaptation problem.

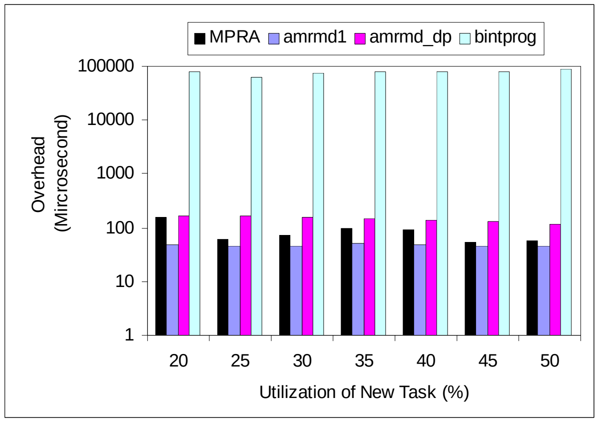


Fig. 5. Admission Control: Online Overhead

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 make the online adaptation in response to the same new task arrivals.

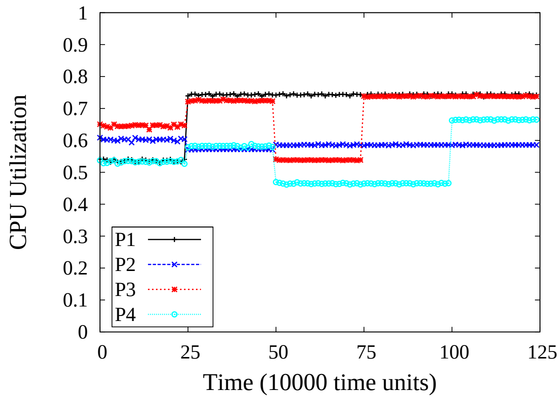
In order to evaluate the efficiency of MPRA, we also investigate its online execution time and compare it with three baselines. The execution times are measured on a 2.52GHz Pentium 4 PC with 1 GB RAM. To achieve fine grained measurements, we use the high resolution timer *gethrtime* provided by ACE [37]. 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.

A. *End-to-end Admission Control*

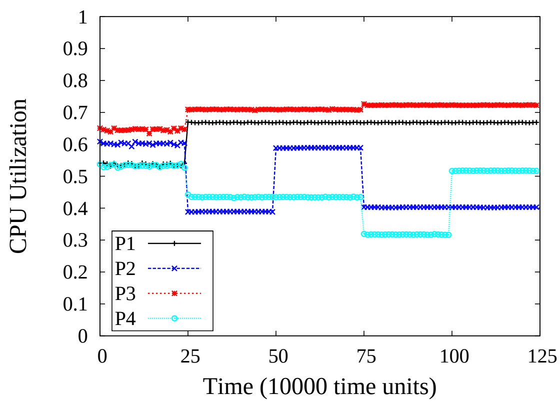
We randomly generated 20 workloads in the simulation of end-to-end admission control. Each workload comprises 8 end-to-end tasks executing on 4 processors. In the following, we present two sets of experiments to evaluate the performance of the four algorithms in the presence of new task arrivals. The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

In the first set of experiments, a new task with utilization of 0.2 is activated at each of the processors at 250000, 500000, 750000, and 1000000 time unit, respectively, which triggers online admission control four times. Figure 3 plots CPU utilizations and the system utility of one run. As seen in Figure 3(a)(b) MPRA, **amrmd1**, and **bintprog** enforce the utilization bounds on all processors by evicting tasks in response to the workload increase. Figure 3(c) shows that MPRA achieves higher system utility than **amrmd1**. MPRA and **bintprog** 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.

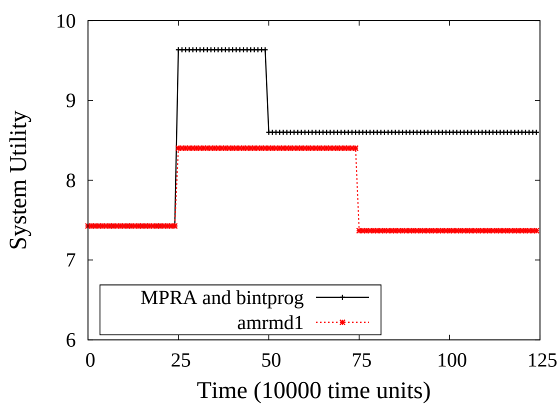
We run the other set of experiments by varying the CPU utilization of the new arrival task from 0.2 to 0.5. Four identical new tasks are activated after 250000 time units on four processors simultaneously. Consequently, online admission control is triggered to maximize system utility while enforcing the utilization bounds. We 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 Figure 4. As shown in Figures 4(a)(b) MPRA consistently achieves higher system utility than both **amrmd1** and **amrmd_dp** under different degrees of workload variations. Moreover, as seen in Figures 4(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



(a) MPRA and bintprog



(b) amrmd1

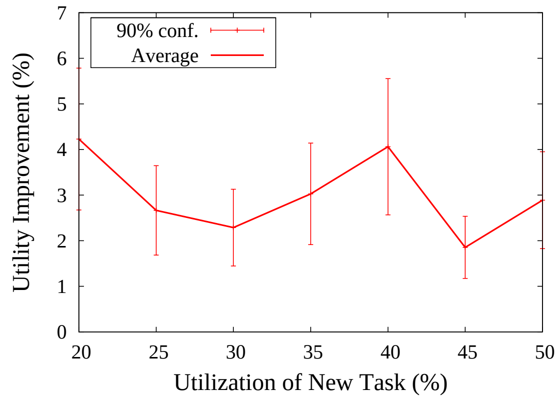


(c) System Utility

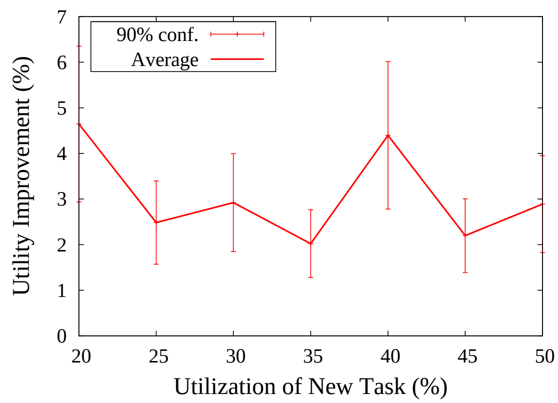
Fig. 6. Rate Adaptation: System Performance under Separate Task Arrivals

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.

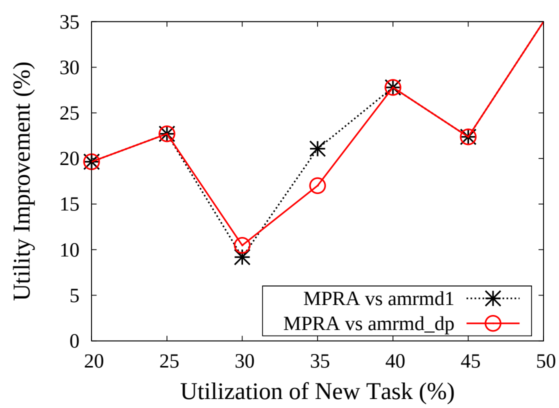
Figure 5 plots the average online execution times of all four algorithms. MPRA, **amrmd1**, and **amrmd_dp** are more than two orders of magnitude faster than **bintprog**. For instance, when the new task has an utilization of 30%, MPRA incurs an overhead of only 100 *microseconds*, while **bintprog** 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.



(a) Average (MPRA vs amrmd1)



(b) Average (MPRA vs amrmd_dp)



(c) Max

Fig. 7. Rate Adaptation: Utility Improvement over Heuristics

B. Discrete Rate Adaptation

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

We use similar sets of experiments as those presented in the previous section to investigate the performance of the four algorithms when applied for discrete rate adaptation. In the first set of experiments, a new tasks arrives at each processor at 250000, 500000, 750000, and 1000000 time unit, respectively. The CPU utilization of each new task is 0.2. As shown in Figure 6, all algorithms maintain acceptable utilizations on all processors in face of new task arrivals. However, both MPRA and **bintprog** generate optimal rates that result in higher system utilities than **amrmd1** in response to new task arrivals.

In the second set of experiments, to generate workload variations, a new task arrives at each of the

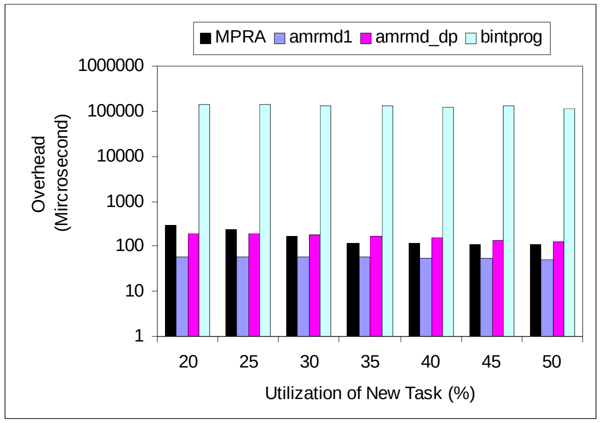


Fig. 8. Rate Adaptation: Online Overhead

four processors simultaneously at 250000 time unit. When new tasks arrive, rate adaptation is triggered to enforce the desired utilization bound and 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** 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 Figure 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.

VI. Conclusions and Future Work

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 as an mp-MILP problem. We then present the design and complexity analysis of MPRA which proves that MPRA can reduce its online complexity to *polynomial* time 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. 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. In the future we plan to extend our work to other online adaptation mechanisms such as task reallocation or dynamic voltage scaling.

In this paper we evaluate the performance of MPRA in response to dynamic task arrivals. Our next step is to explore other workload changes such as execution time variations. In the current implementation MPRA deals with workload changes that can be calculated explicitly. Our approach may be combined with event-driven feedback control to deal with uncertainties in system workload based on measured CPU utilization. The extension is part of our future work.

References

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

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

[3] A. Cervin, J. Eker, B. Bernhardsson, and K.-E. Årzén, "Feedback-Feedforward Scheduling of Control Tasks," *Real-Time Systems*, vol. 23, no. 1-2, pp. 25–53, 2002.

[4] C. Lu, J. Stankovic, G. Tao, and S. Son, "Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms," *Real-Time Systems*, vol. 23, no. 1/2, pp. 85–126, July 2002.

[5] C. Lu, X. Wang, and X. Koutsoukos, "Feedback Utilization Control in Distributed Real-Time Systems with End-to-End Tasks," *IEEE Transactions on Parallel Distributed Systems*, vol. 16, no. 6, pp. 550–561, June 2005.

[6] X. Koutsoukos, R. Tekumalla, B. Natarajan, and C. Lu, "Hybrid Supervisory Utilization Control of Real-Time Systems," in *IEEE RTAS*, 2005.

[7] D. C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole, "A Feedback-driven Proportion Allocator for Real-Rate Scheduling," in *Operating Systems Design and Implementation*, 1999, pp. 145–158.

[8] T. F. Abdelzaher, E. M. Atkins, and K. G. Shin, "QoS Negotiation in Real-Time Systems and Its Application to Automated Flight Control," *IEEE Transactions on Computers*, vol. 49, no. 11, pp. 1170–1183, 2000.

[9] C. Lee, J. P. Lehoczky, D. P. Siewiorek, R. Rajkumar, and J. P. Hansen, "A Scalable Solution to the Multi-Resource QoS Problem," in *IEEE Real-Time Systems Symposium*, 1999, pp. 315–326.

[10] C. Lee, J. Lehoczky, R. Rajkumar, and D. P. Siewiorek, "On Quality of Service Optimization with Discrete QoS Options," in *IEEE Real Time Technology and Applications Symposium*, 1999, pp. 276–.

[11] R. Rajkumar, C. Lee, J. P. Lehoczky, and D. P. Siewiorek, "Practical Solutions for QoS-Based Resource Allocation," in *IEEE Real-Time Systems Symposium*, 1998, pp. 296–306.

[12] C.-G. Lee, C.-S. Shih, and L. Sha, "Online QoS Optimization Using Service Classes in Surveillance Radar Systems," *Real-Time Systems*, vol. 28, no. 1, pp. 5–37, 2004.

[13] J. Acevedo and E. Pistikopoulos, "An Algorithm for Multiparametric Mixed-integer Linear Programming Problems," *Operations Research Letters*, vol. 24, pp. 139–148(10), 1999.

[14] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek, "A Resource Allocation Model for QoS Management," in *IEEE Real-Time Systems Symposium*, Dec. 1997, pp. 298–307.

[15] C. Lee and D. Siewiorek, "An Approach for Quality of Service Management," Technical Report CMU-CS-98-165, Computer Science Department, CMU, 1998.

[16] S. Ghosh, R. R. Rajkumar, J. Hansen, and J. Lehoczky, "Scalable Resource Allocation for Multi-Processor QoS Optimization," in *Proceedings of the 23rd International Conference on Distributed Computing Systems*. Washington, DC, USA: IEEE Computer Society, 2003, p. 174.

[17] H. Tokuda and T. Kitayama, "Dynamic QOS Control based on Real-Time Threads," in *NOSSDAV 93*. London, UK: Springer-Verlag, 1994, pp. 114–123.

[18] S. Brandt, G. Nutt, T. Berk, and J. Mankovich, "A Dynamic Quality of Service Middleware Agent for Mediating Application Resource Usage," in *In IEEE Real-Time Systems Symposium*. Washington, DC, USA: IEEE Computer Society, 1998, p. 307.

[19] S. A. Brandt and G. J. Nutt, "Flexible Soft Real-Time Processing in Middleware," *Real-Time System*, vol. 22, no. 1-2, pp. 77–118, 2002.

[20] X. Wang, D. Jia, C. Lu, and X. Koutsoukos, "Decentralized Utilization Control in Distributed Real-Time Systems," in *IEEE Real-Time Systems Symposium*, 2005.

[21] J. Sun and J. W.-S. Liu, "Synchronization Protocols in Distributed Real-Time Systems," in *International Conference on Distributed Computing Systems*, 1996.

[22] C. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *JACM*, vol. 20, no. 1, pp. 46–61, 1973.

[23] J. P. Lehoczky, "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines," in *IEEE Real-Time Systems Symposium*, 1990, pp. 201–213.

[24] B. Kao and H. Garcia-Molina, "Deadline assignment in a distributed soft real-time system," *IEEE Trans. Parallel Distrib. Syst.*, vol. 8, no. 12, pp. 1268–1274, 1997.

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

[26] T. F. Abdelzaher, K. G. Shin, and N. Bhatti, "Performance guarantees for Web server end-systems: A control-theoretical approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 1, pp. 80–96, 2002.

[27] C. Lu, X. Wang, and C. Gill, "Feedback Control Real-Time Scheduling in ORB Middleware," *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2003.

[28] S. Martello and P. Toth, *Knapsack problems: algorithms and computer implementations*. New York, NY, USA: John Wiley & Sons, Inc., 1990.

[29] O. H. Ibarra and C. E. Kim, "Fast Approximation Algorithms for the Knapsack and Sum of Subset Problems," *J. ACM*, vol. 22, no. 4, pp. 463–468, 1975.

[30] S. Sahni, "Approximate Algorithms for the 0/1 Knapsack Problem," *J. ACM*, vol. 22, no. 1, pp. 115–124, 1975.

[31] T. Gal and J. Nedoma, "Multiparametric Linear Programming," *Management Science*, vol. 18, pp. 406–442, 1972.

[32] M. Kvasnica, P. Grieder, and M. Baotić, "Multi-Parametric Toolbox (MPT)," 2004.

[33] P. Tondel, T. A. Johansen, and A. Bemporad, "Evaluation of Piecewise Affine Control via Binary Search Tree," *Automatica*, vol. 39, pp. 945–950, 2003.

[34] K. G. Murty, "Computational Complexity of Parametric Linear Programming," *Mathematical Programming*, vol. 19, no. 1, pp. 213–219, December 1980.

[35] A. Bemporad, F. Borrelli, and M. Morari, "Model Predictive Control Based on Linear Programming - The Explicit Solution," *IEEE Transactions on Automatic Control*, vol. 47, no. 12, pp. 1974–1985, Dec. 2002.

[36] C. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *Journal of ACM, Vol. 20, No.1, pp. 46-61*, Jan. 1973.

[37] Center for Distributed Object Computing, "The ADAPTIVE Communication Environment (ACE)," www.cs.wustl.edu/∼schmidt/ACE.html, Washington University.