Report Number: WUCSE-2007-15

2007

Optimal Discrete Rate Adaptation for Distributed Real-Time Systems with End-to-End Tasks

Yingming Chen, Chenyang Lu, and Xenofon Koutsoukos

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 changes such as dynamic task arrivals. Through oline preprocessing MPRA transforms a NP-hard utility optimization problem to a set of simple linear functions in different regions expressed in term of CPU utilization changes caused by workload variations. At run time MPRA produces... Read complete abstract on page 2.

Optimal Discrete Rate Adaptation for Distributed Real-Time Systems with End-to-End Tasks

Yingming Chen, Chenyang Lu, and Xenofon Koutsoukos

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 changes such as dynamic task arrivals. Through oline preprocessing MPRA transforms a NP-hard utility optimization problem to a set of simple linear functions in different regions expressed in term of CPU utilization changes caused by workload variations. At run time MPRA produces optimal solutions by evaluating the linear function for the current region. 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.

2007-15

Optimal Discrete Rate Adaptation for Distributed Real-Time Systems with End-to-End Tasks

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 changes such as dynamic task arrivals. Through oline preprocessing MPRA transforms a NP-hard utility optimization problem to a set of simple linear functions in different regions expressed in term of CPU utilization changes caused by workload variations. At run time MPRA produces optimal solutions by evaluating the linear function for the current region. 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 with End-to-End Tasks

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 changes such as dynamic task arrivals. Through offline preprocessing MPRA transforms a NP-hard utility optimization problem to a set of simple linear functions in different regions expressed in term of CPU utilization changes caused by workload variations. At run time MPRA produces optimal solutions by evaluating the linear function for the current region. 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.

1 Introduction

An increasing number of distributed real-time systems operate in dynamic environments where system workload may change at run time. 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. System overload may cause unacceptable delays in important monitoring applications, often at critical times when it is most needed. Such systems must therefore quickly adapt to workload changes. However 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 [18] and meet end-to-end deadlines [19]. 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 [4][6][17][19][8][23], many real-time applications (e.g., avionics [1] and multi-bit video) can only run at a discrete set of predefined rates. Unfortunately, utility optimization problems with discrete options are typically NP-hard [11], especially in the presence of multiple resource constraints. 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 [11][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 [22][11][1][13]. However, these algorithms cannot produce optimal solutions that do not maximize utility.

To overcome the limitations of existing approaches, we present the MultiParametric Rate Adaptation(MPRA) algorithm for discrete rate adaptation. The key novelty and advantage of our approach is that it can *efficiently* produce *optimal* solutions online in face of workload changes. The MPRA algorithm is based on multiparametric mixed-integer linear programming (mp-MILP) [2]. Through offline preprocessing MPRA transforms a NP-hard utility optimization problem to a set of simple linear functions in different regions expressed in term of changes to CPU utilization due to workload changes. At run time MPRA produces optimal solutions by evaluating the linear function for the current region. 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 reduces the online computation complexity of optimal rate adaptation to *polynomial* time, while maximizing system utility in face of workload changes;
- We present simulation results that demonstrate that our algorithm maximize system utility in the presence of dynamic task arrivals, with online execution times two orders of magnitude lower than a standard optimization 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 to demonstrate the optimality and efficiency of the MPRA algorithm. Finally, Section 6 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) [21] for utility optimization in real-time systems. Lee et al. proposed several optimal approaches to the Q-RAM model based on integer programming or dynamic programming [11][10]. These approaches are computationally expensive and unsuitable for online adaptation in distributed real-time systems. To improve the efficiency of the solutions, the authors proposed several efficient heuristic algorithms that can only produce sub-optimal solutions [22][11][10]. While the heuristic proposed in [10] can generate solutions with a bounded distance from optimal solutions for the single resource case [12], there does not exist an analytical bound for the heuristic algorithms for the multiple resource case, which are common in distributed real-time systems. Abdelzaher et al. also developed a QoS-negotiation model called RTPOOL and a heuristic algorithm to improve system utility [1]. Recently, Lee et al. introduced a method called service class configuration to address the online adaptation problem with dynamic arrival and departure of tasks [13]. 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 techniques have been proposed to enforce real-time performance in real-time systems. For example, several feedback control scheduling algorithms [6][4][17][23] were designed to control the performance of a *single* processor by adjusting task rates, while several other control-theoretic approaches were designed for scheduling in distributed real-time systems [19][26]. All the above solutions assume that task rates can be adjusted in a continuous range. While this assumption holds for certain classes of systems, there are many systems, such as avionics and total-ship computing environments that only support a finite a priori set of discrete task rates. In contrast, our work focuses on distributed real-time systems with *discrete* task rates. HySUCON [8] is a heuristic algorithm for scheduling real-time systems that support discrete task rates. However, it is only applied to single processor systems and cannot produce optimal solutions. There are several important differences between our work and earlier work on rate adaptation. First, our work deals with distributed real-time systems with *discrete* task rates, while earlier work (except HySUCON) cannot handle discrete rates in distributed real-time systems. Second, none of the aforementioned projects is designed to maximize system utility in distributed real-time systems. In addition, we assume the CPU utilization changes are known to the system when workload variations occur. For example, the system knows the requested utilization of a new task at its arrival time. As a result, the system does not need *feedback* control. We note that our approach may be combined with event-driven feedback control to deal with uncertainties in system workload. The extension is part of our future work.

3 Problem Formulation

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

3.1 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 is also periodic and shares the same rate as T_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 defined to be the utility value contributed by T_i when

it is configured with $r_i^{(j)}$. We assume each task can be evicted, i.e., $r_i^{(0)} = 0, 1 \leq i \leq m$. Note that 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.

3.2 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 $r_i \in R_i, 1 \leq i \leq m$.
- D: $D = [d_1, ..., d_n]$ is the workload change vector where d_i is the change to the utilization of the i^{th} processor caused by workload variations. D may be caused by a variety of sources such as arrivals and departures of critical tasks that must be executed at fixed rate.
- U: $U = [u_1, ..., u_n]$ is the CPU utilization vector where u_i represents the utilization of the i^{th} processor in the system. The relationship between u_i and d_i is given 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 utilization bounds are used to enforce the resource constraints in distributed real-time systems via setting $U \leq B$. The utilization bound may be set to the appropriate schedulable utilization bound to meet end-to-end deadlines[1] or a threshold for overload protection.
- Q_s: Q_s is the system utility, which is defined to be the sum of the task utilities, i.e., $Q_s = \sum_{i=1}^{m} q_i$ where q_i is the current task utility of T_i and $q_i \in Q_i$. This can be easily extended to other linear functions.

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.

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

subject to two sets of constraints

$$U \leq B \tag{2}$$

[1]In the end-to-end scheduling approach [24], the deadline of an end-to-end task is divided into subdeadlines of its subtasks. A well-known approach for meeting the subdeadlines on a processor is by enforcing an appropriate schedulable utilization bound [15][14].

$$r_i \in R_i, 1 \leq i \leq m \tag{3}$$

The utilization constraint (2) ensures that no processor exceeds its utilization bound. The constraints (3) indicate that each task can only be configured with predefined rates.

The discrete rate adaptation problem can be easily reduced to the 0-1 Knapsack Problem, which is know to be NP-hard [20]. This indicates that the discrete rate adaptation problem is also NP-hard. It is therefore impractical to apply standard optimization approaches to discrete rate adaptation in distributed real-time systems.

4 MultiParametric Rate Adaptation (MPRA) Algorithm

In this section, we present the design and analysis of MPRA for discrete rate adaptation in distributed real-time systems. We first give a brief overview of the general theories of multiparametric programming. Next, we transform the discrete rate adaptation problem to a mp-MILP problem, which allows us to design MPRA that instantiates the multiparametric programming approach to *efficiently* produce *optimal* rates in response to workload variations in distributed real-time systems. Finally, we present the complexity analysis of our algorithm.

4.1 Overview of Multiparametric Programming

Multiparametric programming is an approach for solving mathematical programming problems with constraints that depend on varying parameters [7]. The multiparametric programming approach includes an offline and an online component. The offline component partitions the space of varying parameters into critical regions. For each critical region, the objective and optimization variables are expressed as linear functions of the parameters. For a given value of the varying parameter, the online component computes the solution by evaluating the explicit function for the critical region which includes the parameter value.

The multiparametric approach has been extended for multiparametric mixed-integer linear programming problems (mp-MILP) [2]. The algorithm presented in [2] 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

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

where $\Theta_i \stackrel{\triangle}{=} \{\theta \in \Xi : H_i\theta \leq k_i\}, i = 1, ..., N_r$ are a partition of the space of varying parameters. N_r is the number of critical regions generated by the mp-MILP algorithm. For each θ, an unique region i which includes θ can be located and the optimization vector x can be computed by evaluating $x = P_i\theta + q_i$.

We observe that mp-MILP approaches are suitable for distributed real-time systems that must handle workload changes by switching among discrete task rates. 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.

4.2 Problem Transformation

In this subsection, we show how to transform the discrete rate adaptation problem presented in Section 3.2 to an mp-MILP problem. We start with the end-to-end admission control problem, which is a special case of discrete rate adaptation.

4.2.1 End-to-end Admission Control

In this special case, 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

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

We introduce a $n \times m$ matrix F, where $f_{ij} = \sum_{T_{jl} \in S_{ij}} r_j^{(1)} c_{jl}$, and $f_{ij} = 0$ if no subtask of T_j is allocated on processor P_i. The relationship between U, D and X can be described by the following equation:

$$U = D + FX \quad (10)$$

We assume that the task utility contributed by T_i is zero when it is evicted, i.e., $q_i^{(0)} = 0$. So 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. The system utility will be $Q_s = \sum_{1 \leq i \leq m} q_i^{(1)} x_i$. 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:

$$\min_X \sum_{1 \leq i \leq m} -q_i^{(1)} x_i \quad (11)$$

subject to

$$FX \leq D_N \quad (12)$$

$$x_i \in \{0, 1\}, 1 \leq i \leq m \quad (13)$$

4.2.2 Discrete Rate Adaption

We first introduce a rate adaptation vector X with $\sum_{1 \leq i \leq m} k_i$ elements to represent the rate configuration of the system such that

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

where $1 \leq i \leq m$, $1 \leq j \leq k_i$, and $l = \sum_{1 \leq t < i} k_t + j$. The task rate vector R can be obtained by $R = WX$, where W is a $m \times (\sum_{1 \leq i \leq m} k_i)$ matrix such that

$$w_{il} = \begin{cases} r_i^{(j)} & \text{if } \sum_{1 \leq t < i} k_t < l \leq \sum_{1 \leq t \leq i} k_t \\ 0 & \text{otherwise} \end{cases} \quad (15)$$

where $1 \leq i \leq m$, $1 \leq l \leq \sum_{1 \leq t \leq m} k_t$, and $j = l - \sum_{1 \leq t < i} k_t$.

We then introduce a $n \times m$ matrix H, where $h_{ij} = \sum_{T_{jl} \in S_{ij}} c_{jl}$ 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

$$U = D + HWX \quad (16)$$

To describe the relationship between Q_s and X, we introduce a vector $\bar{Q}$ such that $\bar{q}_l = q_i^{(j)}$ where $1 \leq i \leq m$, $1 \leq j \leq k_i$, and $l = \sum_{1 \leq t < i} k_t + j$. Thus, the system utility can be computed using the following equation $Q_s = \bar{Q}X$. By denoting $D_N = B - D$ and $G = HW$, we re-formulate the discrete rate adaptation problem as following:

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

subject to

$$GX \leq D_N \quad (18)$$

$$x_i \in \{0, 1\}, 1 \leq i \leq \sum_{1 \leq j \leq m} k_j \quad (19)$$

$$\sum_{\sum_{1 \leq t < i} k_t < j \leq \sum_{1 \leq t \leq i} k_t} x_j \leq 1, 1 \leq i \leq m \quad (20)$$

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 formulation.

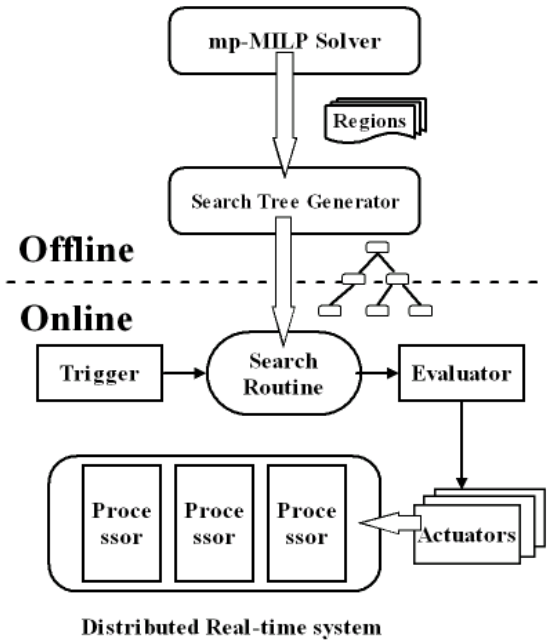


Figure 1. General Overview of MPRA Algorithm

4.3 Design of MPRA

After transforming the discrete rate adaptation problem to an mp-MILP problem, we present the MPRA algorithm to solve it. Based on mp-MILP approaches, our algorithm can produce *optimal* rate adaptation solutions online in response to workload changes. As shown in Figure 1, MPRA has both offline part and online part. The offline part consists of an mp-MILP Solver and a Search Tree Generator. The online part is composed of a Trigger, a Search Routine, an Evaluator, and multiple Actuators.

The offline part only executes once before the system starts running. It first invokes the mp-MILP Solver to divide the n-dimensional space of D_N into multiple regions and generate the explicit linear function which expresses X as a linear function of D_N for each region. It then calls the Search Tree Generator to build a binary tree for the representation of those regions. This binary tree will be used by the online part of MPRA .

The online part is invoked when workload changes occur at run time. It works as follows: (1) the Trigger sends the current value of D to the Search Routine; (2) the Search Routine goes through the binary tree and finds the region which the current value of D_N belongs to; (3) the Evaluator computes the new value of X and sends it to Actuators; finally, (4) Actuators change task rates according to the new value of X.

In the following, we present the functionality of each component in detail.

4.3.1 Offline Components

The offline part of MPRA includes following two components:

- **mp-MILP Solver**: It generates the explicit optimal solution, which is a PWA function with a polyhedral partition of the space of D_N, for a given mp-MILP problem. Our implementation of MPRA uses an mp-MILP solver provided by the MultiParametric Toolbox (MPT) [9], which implements the mp-MILP algorithm presented in [2].

- **Search Tree Generator**: It generates a binary tree data structure for the representation of the explicit solution generated by the mp-MILP Solver. Based on the binary tree, the time of the online rate adaptation operation becomes logarithmic in the number of regions. MPRA uses the binary search tree generator provided by the MPT toolbox [9], which implements the algorithm presented in [25].

4.3.2 Online Components

The online part of MPRA comprises the following components:

- **Trigger**: Rate adaptation is only triggered by workload variations, such as new task arrivals and task departures. When workload changes occur, the Trigger starts an adaptation operation and sends the current value of D to the Search Routine. For example, when a critical task arrives at run time, the Trigger passes the CPU utilization of the new task to the Search Routine to start rate adaptation.

- **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.

- **Evaluator**: The Evaluator computes the new value of X by evaluating the explicit linear function of the region located by the Search Routine. It then sends the new value of X to Actuators.

- **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.

4.4 Complexity Analysis

In this section we analyze the complexity of the online computation of MPRA when applied to the

discrete rate adaptation problem. Our analysis focuses on the online search routine and the evaluation of the explicit solution, which dominate the online rate adaptation algorithm.

The complexity of the online search routine depends on N_r, the number of critical 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 [2]. It recursively fixes the elements in X and builds an enumeration tree. There will be $2^{\bar{m}-1}$ leaf nodes in the tree, where $\bar{m} = \sum_{1 \le i \le m} k_i$. Let $k = \max\{k_1, ..., k_m\}$. Then $\bar{m} \le km$. For each leaf node, $\bar{m} - 1$ elements in X have been fixed and the problem is relaxed to a multiparametric linear programming problem (mpLP) by considering the $\bar{m}^{th}$ element as a continuous variable in [0,1]. Based on the results in [3], the upper bound to the number of critical regions for one leaf node can be obtained by $n_r \le n + 1$, where n is the number of processors.

The solution of the mpLP problem for each leaf node is feasible but it may not be optimal to the original problem, because the critical regions for different leaf nodes can be overlapping with each other. The optimal solution of the original problem can be obtained by removing the overlap among these regions. 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 will get N_r non-overlapping regions. N_r is bounded by

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

All N_r non-overlapping regions together represent a partition of the entire space of D_N. The explicit solution of each non-overlapping region is optimal to the original problem. Then we can obtain a PWA function with this partition, which represents the optimal solution for the original problem.

The binary tree generated by the Search Tree Generator is helpful for reducing the complexity of online region search. The Generator constructs a tree such that for a given D_N we only evaluate one linear inequality at each level. For one linear inequality evaluation we do 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 [25], the depth of the binary tree, d, is given by

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

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 [25]. So the worst-case number of arithmetic operations required for online search and evaluation is $(2n+1)d + 2\bar{m}n$, i.e., MPRA has time complexity $O(mn)$.

Therefore the online complexity of MPRA is *polynomial* in the number of tasks and the number of processors for the discrete rate adaptation problem.

5 Evaluation

In this section, we present simulation results for both end-to-end 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 preprocessing of MPRA is done in MATLAB.

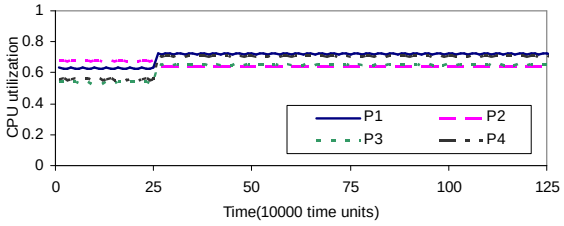
In our simulation, the subtasks on each processor are scheduled by the Rate Monotonic scheduling (RMS) algorithm [16]. We assume that each task's end-to-end deadline $d_i = m_i/r_i(k)$, where m_i is the number of subtasks of task T_i. We then evenly divide the deadline into subdeadlines for its subtasks. The resultant subdeadline of each subtask T_{ij} equals to its period, $1/r_i(k)$. Hence we choose the schedulable utilization bound of RMS [16] 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. In our experiments, online adaptation operations are triggered by new task arrivals.

5.1 Baselines

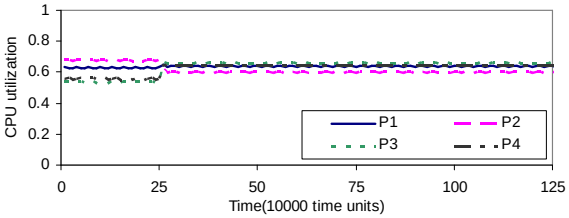
In order to evaluate the optimality and efficiency of MPRA, we compare it with two existing algorithms: **bintprog** and **amrmd1**. **bintprog** is a binary integer linear programming solver provided by the commercial Optimization Toolbox from MATLAB 7. **bintprog** is a representative optimization solver that can produce optimal solutions, which is used to verify the optimality of MPRA. On the other hand, **amrmd1**, which stands for Approximate Multi-Resource Multi-Dimensional Algorithm, is a representative efficient heuristic algorithm that can only produce suboptimal solutions [11]. These two algorithms can be easily applied for discrete rate adaptation in distributed real-time systems.

5.2 End-to-end Admission Control

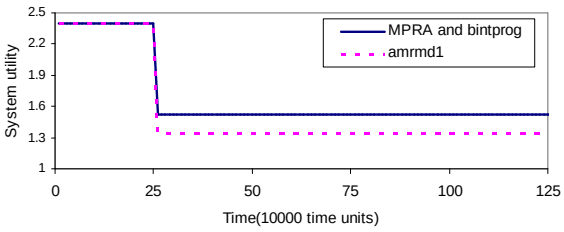
We use a workload that comprises 8 end-to-end tasks (with a total of 21 subtasks) executing on 4 processors. In the following, we present three sets of simulations to evaluate the performance of the three algorithms in the presence of new task arrivals. Such new tasks are viewed as mission critical tasks that must be executed at the cost of other tasks. We emulate them using a highest-priority periodic task to compete with the tasks in our workload for CPU resource on each processor. In experiment I, four new tasks arrive simultaneously. In experiment II, four



(a) CPU Utilization(MPRA and bintprog)

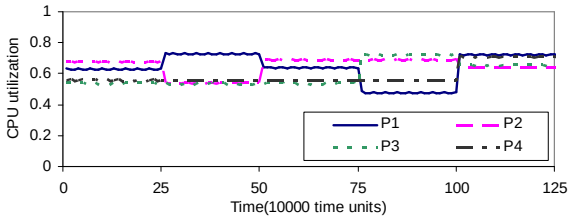


(b) CPU Utilization(armrmd1)

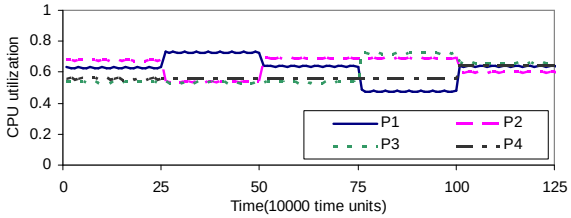


(c) Utility

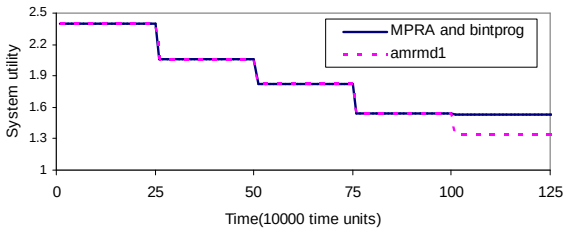
Figure 2. Admission control: system performance under simultaneous task arrivals



(a) CPU Utilization(MPRA and bintprog)



(b) CPU Utilization(armrmd1)



(c) Utility

Figure 3. Admission control: system performance under separate task arrivals

new tasks are activated sequentially. In experiment III, we compare the three algorithms under different sizes of new tasks.

Experiment I: Simultaneous Task Arrivals We now evaluate the three approaches in response to simultaneous task arrivals. In this experiment, each new arrival task invokes a function with an execution time of 60 time units, every 200 time units. Four new tasks are activated after 250000 time units on four processors simultaneously. Consequently, online admission control is triggered to guarantee the execution of new tasks while maintaining resource constraints and maximizing system utility. Figure 2 shows that all three approaches respond to this event by evicting tasks. With the help of online admission control, all CPU utilizations remain less than their utilization bounds in face of new task arrivals. We note that MPRA and **bintprog** produce the same optimal solutions and hence achieve the same system utility in all the experiments presented in this paper. This confirms that the solutions generated by MPRA are optimal. Thus, we always show the performance results of these two approaches together. Another important observation is that MPRA achieves 15% improvement in system utility when compared to **amrmd1** after admission control as shown in Figure 2(c).

Experiment II: Separated Task Arrivals In this experiment, four new tasks are activated sequentially on four processors after 250000, 500000, 750000, and 1000000 time units, which sequentially trigger the online admission control four times. Figure 3 shows that all three approaches handle these four task arrivals immediately. After four new tasks arrive, the system utilities achieved by MPRA and **amrmd1** are 1.53 and 1.34, respectively, which shows MPRA achieves higher system utility than **amrmd1** does.

Experiment III: Varying Utilization of New Arrival Tasks To further compare the performance of the three algorithms, we run a set of experiments by varying the CPU utilization of the new arrival task from 0.2 to 0.45. We plot the system

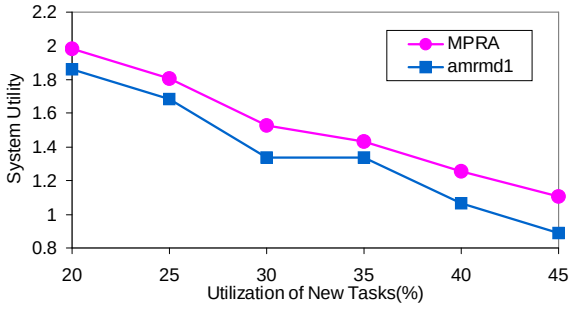


Figure 4. Admission control: system performance under varying utilization of new arrival tasks

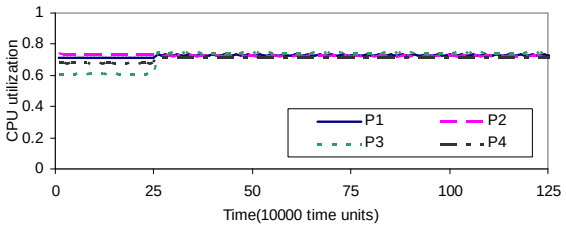
utilities achieved by MPRA and **amrmd1** against different utilizations of new tasks in Figure 4. Every data point is based on the system utility achieved after the online admission control operations. MPRA consistently achieves higher system utility than **amrmd1** does under different degrees of utilizations of new tasks.

5.3 Discrete Rate Adaptation

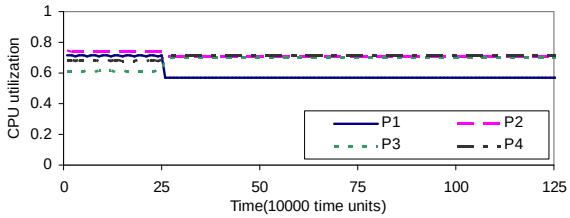
We use a workload that includes 6 end-to-end tasks (with a total of 17 subtasks) executing on 4 processors. All tasks can be dynamically evicted. Each task has two non-zero rate options. We use the same sets of experiments presented in the previous section to investigate the performance of the three algorithms when applied to the discrete rate adaptation problem.

Experiment I: Simultaneous Task Arrivals In this experiment, we use a new arrival task on each processor that invokes a function with a fixed execution time of 45 time units, every 100 time units. Four identical new tasks arrive after 250000 time units simultaneously. When new tasks arrive, rate adaptation is triggered to maintain the required real-time performance and maximize system utility. The results in Figure 5 demonstrate that MPRA and **bintprog** perform 25% better than **amrmd1** does with respect to system utility after rate adaptation.

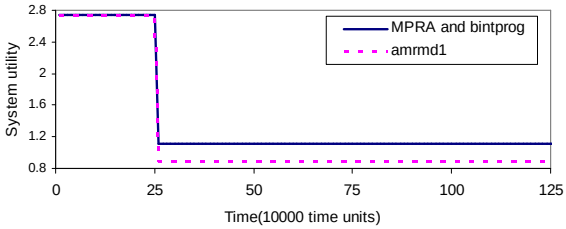
Experiment II: Separated Task Arrivals In this experiment, four new tasks arrive sequentially on four processors after 250000, 500000, 750000, and 1000000 time units. The CPU utilization of each new task is 0.3. As shown in Figure 6, while all algorithms maintain acceptable utilizations on all processors in face of new task arrivals at run time, MPRA generates optimal solutions and produces higher system utilities in response to each new task arrival than **amrmd1**



(a) CPU Utilization(MPRA and bintprog)



(b) CPU Utilization(armrmd1)



(c) Utility

Figure 5. Rate adaptation: system performance under simultaneous task arrivals

does.

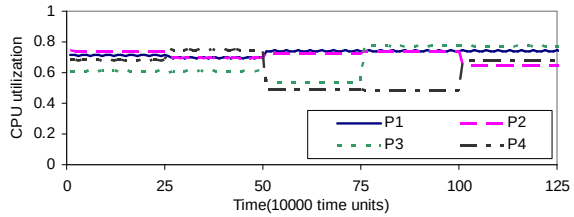
Experiment III: Varying Utilization of New Arrival Tasks Figure 4 plots the system utilities achieved by MPRA and **amrmd1** against the CPU utilization of the new task, which varies from 0.2 to 0.45. While **amrmd1** achieves the same utility as MPRA when the utilization of the new task is 0.4, MPRA performs better than **amrmd1** with respect to system utility in all the other cases.

5.4 Run-Time Overhead

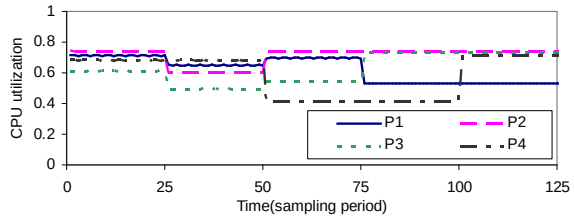
In this section we show the online execution times of the three solutions to compare their run-time efficiency. We used a 2.52GHz Pentium 4 PC with 1 GB RAM for performance evaluation. To achieve fine grained measurements, we adopt a nanosecond scale time measuring function called gethrtime provided by the ACE environment [5]. This function uses an OS-specific high-resolution timer that returns the number of clock cycles since the CPU was powered up or reset. The gethrtime function has a low overhead and is based on a 64 bit clock cycle counter on Pentium

Experiment	MPRA (ms)	bintprog(ms)	amrmd1(ms)
Admission Control I	0.41	148.98	0.041
Admission Control II	0.13	73.90	0.041
Rate Adaptation I	1.13	183.83	0.055
Rate Adaptation II	0.69	149.07	0.056

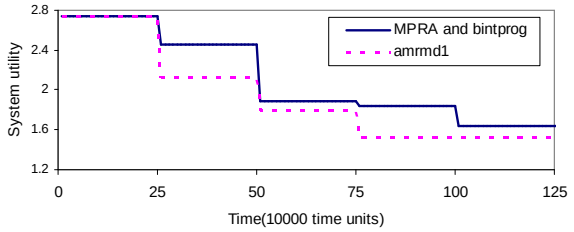
Table 1. Overhead



(a) CPU Utilization(MPRA and bintprog)



(b) CPU Utilization(armrmd1)



(c) Utility

Figure 6. Rate adaptation: system performance under separate task arrivals

processors. With the clock counter number divided by the CPU speed, we can get reasonably precise and accurate time measurements. 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. The overhead of the three approaches are summarized in Table 1. Both MPRA and **amrmd1** are significantly faster than **bintprog**. While MPRA incurs higher overhead than **amrmd1**, we expect that its overhead is acceptable in many distributed real-time systems. The results show that MPRA can solve the discrete rate adaptation problem with low online execution times.

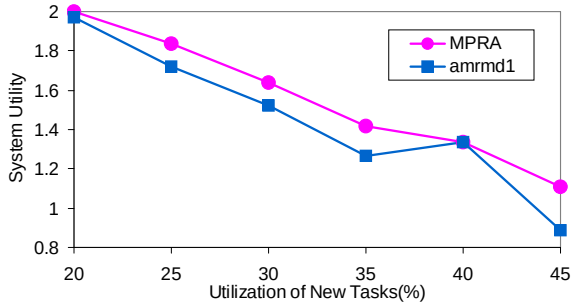


Figure 7. Rate adaptation: system performance under varying utilization of new arrival tasks

6 Conclusions

We have developed the MPRA algorithm for optimal and efficient discrete rate adaptation in distributed real-time systems. In this paper, we first formulate 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 online execution times more than two orders of magnitude lower than a standard optimization solver. MPRA also achieves up to 25% improvement in system utility when compared to the **amrmd1** algorithm with acceptable online overhead. Although we focus on the discrete rate adaptation problem in this paper, our methodology may be extended to other online reconfiguration problems with discrete options.

References

[1] 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*, 49(11):1170–1183, 2000.

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

[3] A. Bemporad, F. Borrelli, and M. Morari. Model Predictive Control Based on Linear Programming -

The Explicit Solution. *IEEE Transactions on Automatic Control*, 47(12):1974–1985, Dec. 2002.

[4] G. C. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni. Elastic Scheduling for Flexible Workload Management. *IEEE Trans. Comput.*, 51(3):289–302, 2002.

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

[6] A. Cervin, J. Eker, B. Bernhardsson, and K.-E. Årzén. Feedback-Feedforward Scheduling of Control Tasks. *Real-Time Systems*, 23(1-2):25–53, 2002.

[7] T. Gal and J. Nedoma. Multiparametric Linear Programming. *Management Science*, 18:406–442, 1972.

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

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

[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*, pages 276–, 1999.

[11] 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*, pages 315–326, 1999.

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

[13] C.-G. Lee, C.-S. Shih, and L. Sha. Online QoS Optimization Using Service Classes in Surveillance Radar Systems. *Real-Time Systems*, 28(1):5–37, 2004.

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

[15] C. Liu and J. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *JACM*, 20(1):46–61, 1973.

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

[17] C. Lu, J. Stankovic, G. Tao, and S. Son. Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms. *Real-Time Systems*, 23(1/2):85–126, July 2002.

[18] 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.

[19] 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*, 16(6):550–561, June 2005.

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

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

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

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

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

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

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