A Metric and Mixed-Integer-Programming-Based Approach for Resource Allocation in Dynamic Real-Time Systems

Sethavidh Gertphol, Yang Yu, Shriram B. Gundala, Viktor K. Prasanna
University of Southern California
Department of EE-Systems
{gertphol, yangyu, gundala, prasanna}@usc.edu

Shoukat Ali, Jong-Kook Kim
Purdue University
School of Electrical and Computer Engineering
{alis, kim42}@ecn.purdue.edu

Anthony A. Maciejewski♯, and Howard Jay Siegel♯†
Colorado State University
♯Department of Electrical and Computer Engineering
†Department of Computer Science
{aam, hj}@colostate.edu

Abstract

Dynamic real-time systems such as embedded systems operate in environments in which several parameters vary at run time. These systems must satisfy several performance requirements. Resource allocation on these systems becomes challenging because variations of run-time parameters may cause violations of the performance requirements. Performance violations result in the need for dynamic re-allocation, which is a costly operation. In this paper, a method for allocating resources such that the allocation can sustain the system in the light of a continuously changing environment is developed. We introduce a novel performance metric called $MAIL$ (maximum allowable increase in load) to capture the effectiveness of a resource allocation. Given a resource allocation, $MAIL$ quantifies the amount of additional load that can be sustained by the system without any performance violations. A Mixed-Integer-Programming-based approach (MIP) is developed to determine a resource allocation that has the highest $MAIL$ value. Using simulations, several sets of experiments are conducted to evaluate our heuristics in various scenarios of machine and task heterogeneities. The performance of MIP is compared with three other heuristics: Integer-Programming based, Greedy, and classic Min-Min. Our results show that MIP performs significantly better when compared with the other heuristics.

1 Introduction

Dynamic real-time systems such as embedded systems [6] have recently gained importance in several application domains, such as automobile control, avionics, and defense. These systems are comprised of sensors, actuators, and processing nodes that execute various application tasks. These systems operate in an environment that continuously changes. The changes in the environment are reflected in several parameters that vary at run-time. The system detects the changes in the environment through sensors and, after certain computations, reacts to it via actuators. The system must react in a timely manner to changes. Violating this might cause catastrophic effects on the system.

System resources, such as computing power and communication channel bandwidth, must be allocated efficiently for the system to react timely to changes in the environment. Thus, the primary goal of any resource allocation should be the allocation of suf-

This research was supported by the DARPA/ITO Quorum Program through the Office of Naval Research under Grant No. N00014-00-1-0599.

0-7695-1573-8/02/$17.00 (C) 2002 IEEE

ficient resources to each task or group of tasks, so that they complete their execution within the specified deadlines. Several algorithms and heuristics (e.g., [2, 3, 12, 18, 20]) have been developed to allocate resources using system parameters that are known a priori. However, they do not consider run-time parameter variations (e.g., the amount of input data) that may cause violations in performance requirements. To sustain the system in its current status, dynamic re-allocation is often used to re-assign resources to tasks. However, determining a new allocation and performing re-allocation consumes time and resources. Because re-allocation is usually needed when tasks lack adequate resources, an overhead is incurred when the system is least capable to afford it. Thus, dynamic re-allocation is a costly operation with respect to resource utilization (especially computation time). Allocation algorithms that do not consider the effects of run-time parameter variations on task execution time may have to pay the heavy price of early re-allocation.

Task allocation in real-time systems to meet certain deadlines is known to be an NP-hard problem [11, 14]. The problem becomes more difficult when variations in run-time parameters must also be taken into account. To evaluate the performance of such an allocation, a performance metric that captures such variations is needed. However, quantifying such a metric is difficult due to the unpredictable nature of the parameter variations.

In this paper, we propose a method for allocating resources such that the allocation can sustain the system in the light of continuously changing environment. In doing so, more system resources are available to application tasks for execution, rather than being used for performing re-allocation. We make the following contributions in this paper:

- A novel performance metric that is suitable for evaluating resource allocation in systems that operate in a changing environment is proposed (Section 3). A high-level abstraction of the run-time parameters, their variations, and their effects on performance attributes of the system is used to determine the "quality" of a given allocation.
- System and application models (Section 4.1 and 4.2) that take into account the variation in run-time parameters are developed. Based on these models, the performance metric is formalized in Section 4.3. An objective function that leads to a good allocation under some assumptions is discussed (Section 5.2).
- The performance of our Mixed-Integer-Programming-based heuristic (MIP) is compared with three other heuristics (Integer-Programming-based (IP), Greedy, and Min-Min) [8] in Section 6. Simulations show that MIP achieves up to *two times* performance improvement over the other heuristics.

The rest of this paper is organized as follows. A brief discussion of related work is presented in Section 2. Section 3 describes an abstract representation of the performance metric. Section 4 discusses the system, application, and execution models. A formal mathematical formulation of the problem based on the models is discussed in Section 5. Simulation results are analyzed in Section 6. Concluding remarks are presented in Section 7.

2 Related Work

Many research efforts in the literature concentrate on real-time resource allocation and scheduling problems in both uniprocessor (e.g., [2, 3, 20]) and multiprocessor environments (e.g., [5, 9, 13, 17]). Most of them cannot be directly applied to our work because our research assumes an environment that includes multitasking on multiple machines with multitasking communication links, continuously running applications, and heterogeneous distributed systems. Also, most of these research efforts focus on finding a feasible allocation or an optimal allocation in the sense of some *static* performance metrics. In our research, we define our performance metric to capture the variation of run-time parameters.

The work in [4, 10, 16] has a similar application model for sensors, tasks, and actuators as the model used in this paper. These research efforts are different because in our research the processors and the communication links can perform multitasking and the applications that need to be allocated execute continuously.

The system model assumed in our research is similar to that of the DeSiDeRaTa project (e.g., [19]) in that both use continuously running applications and a heterogeneous distributed system. The difference is that, while our work focuses on deriving an efficient allocation of resources to tasks, the DeSiDeRaTa project focuses on dynamic re-allocation of resources to meet the real-time constraints.

3 Performance Metric

A dynamic real-time system has several parameters that vary at run time. A "robust" resource allocation has to satisfy all the performance constraints of the system by considering the impact of run-time parameter

0-7695-1573-8/02/$17.00 (C) 2002 IEEE

variations. In this section, we propose a performance metric that can measure how well-prepared a dynamic real-time system is for absorbing the run-time variations in the environment without violating performance constraints.

Consider a dynamic real-time system with n run-time parameters that characterize the current status of the environment. An n-dimensional system state space where each dimension corresponds to a run-time parameter can be defined. Each point in the state space, denoted as an operating point, represents a system state. This state is determined by the corresponding values of run-time parameters. A set of performance attributes is associated with each operating point. Given a resource allocation x, these attributes for each point can be calculated from the corresponding coordinates of that point. The system must satisfy some specified performance constraints. An operating point is said to be valid if all these constraints are satisfied by the performance attributes of this point. For the sake of discussion, all the valid operating points are assumed to form a closed and continuous space, called an operating region. Given a resource allocation and performance constraints, an operating region can be defined on the state space.

An initial operating point is specified by the initial values of all run-time parameters. This initial operating point is the same for all allocations. An allocation is feasible iff the initial operating point lies within its operating region. Throughout this section, only feasible allocations are considered. The system transits from this initial operating point when variations in run-time parameters occur. When the system transits out of the operating region of an allocation, a resource re-allocation is needed. The operating region of this new allocation must cover the current operating point of the system. The general goal of our allocation is to delay the *first re-allocation* of resources due to run-time variations in the environment. Therefore, the goal is to find an allocation whose operating region covers, for the longest period of time, all the operating points to which the system may transit during run time.

Let R denote an operating point, with R_0 denoting the initial operating point. Let $P(R, x)$ denote the vector of performance attributes given an operating point R and an allocation x. Let d_R denote a norm on the space of values specified by run-time parameters; and d_P, a norm on the space of performance attributes. Intuitively, the highest value of $d_P(P(R, x), P(R_0, x))$, where R represents a valid operating point, can be used to measure the "quality" of an allocation x. However, it is desirable to characterize the "quality" of an allocation in terms of the variation of the input parameters (run-time parameters). Thus, a good allocation should have the highest value of $d_R(R, R_0)$, where R represents a valid operating point. We shall denote this performance metric as the maximum allowable variation in run-time parameters. Because every allocation corresponds to an operating region, it has its own maximum allowable variation in run-time parameters. The goal of our allocation problem is thus to find an allocation that has the highest value of this maximum allowable variation in run-time parameters.

4 Resource Allocation Problem

4.1 System Model

We consider a system consisting of s multitasking-capable machines, represented by a set $\mathcal{M} = \{m_1, \ldots, m_s\}$. Each machine is connected to a network switch via a full-duplex communication link. The capacity of the communication links may be different.

4.2 Application Model

As an example of dynamic real-time systems, we consider a sensor-actuator network in this paper. Such a system (shown in Figure 1) consists of sensors, actuators, and processing tasks. The sensors continuously send information about the environment to the tasks. These tasks process the data from the sensors and issue commands to the actuators. Such systems are widely used in various fields.

The set of processing tasks in such systems is modeled using the asynchronous dataflow (ASDF) process network [15]. Each task in the ASDF process network has associated with it a firing rule, which determines when a task starts execution based on the availability of its inputs. However, the ASDF process network does not specify any real-time requirements or parameters that can vary at run time. We extend the ASDF process network to capture the variations of run-time parameters as well as to associate real-time requirements with a task or a group of tasks.

Let $\mathcal{S}$ denote the set of sensors, and $\mathcal{T}$ denote the set of actuators. Sensors and actuators are part of the system hardware, and thus cannot be allocated. An ASDF process network, denoted as $\mathcal{G}(\mathcal{E}, \mathcal{A})$, where $\mathcal{E}$ is a set of edges and $\mathcal{A}$ is a set of nodes, is used to model the set of processing tasks. $\mathcal{A}$ represents the set of tasks to be allocated, and each edge in $\mathcal{E}$ connecting two nodes represents a directed data flow from one task to the other task.

A. Load levels

For every processing node that receives data, exactly one incoming edge is set to be a primary edge (dark, solid edges in Figure 1). In this study, ASDF process networks with cycles consisting of primary edges only are not considered. Let $\mathcal{E}_P$ be the set of all primary edges. The amount of data that a task receives from its primary edge can vary during run time. Load level of task a_i, $LL(a_i)$, is a scalar value that represents the data that task a_i receives through its primary edge. Let vector LL denote the load level of all tasks, with LL_{init} denoting the initial load level. Data arriving from non-primary edges (gray, dashed edges in Figure 1) is used for information updating (e.g., updating the internal database of the task). This is considered to consume some CPU cycles independent of the load level of a task. A task starts execution only after it receives data from its primary edge. In the ASDF terminology, the firing rule of a task can be stated as: a task fires (starts execution) only when data is received from the primary edge.

B. Load level variation

The changes in the load level of a task are reflected in the variation of its computation and communication latencies. An estimated-time-to-compute-function matrix, $ETCF$, gives a computation latency function for each task-machine pair (a_i, m_l) that maps $LL(a_i)$ to an estimated computation time of a_i on m_l. The size of the $ETCF$ matrix is $|\mathcal{A}| \times |\mathcal{M}|$. For this study, each entry in the $ETCF$ matrix, $ETCF[a_i, m_l]$, is assumed to be a linear function of the load level on the primary edge of task a_i, with a positive slope, $ETCF[a_i, m_l].a$, and a non-negative intercept, $ETCF[a_i, m_l].b$. The slope determines the rate at which the estimated computation latency varies with respect to the variation of the load level on the primary edge of the task during run time. The intercept determines the CPU time overhead for a task to process the data arriving from its non-primary edges. Because we are interested in those cases where the load levels on the primary edges determine the performance of the system, in this study, the intercept is assumed to be zero for each task. An estimated-time-to-c(K)ommunicate-function matrix, $ETKF$, gives a function for each task-machine pair (a_i, m_l) that maps $LL(a_i)$ to an estimated communication time for task a_i from machine m_l to the network switch. The size of the $ETKF$ matrix is also $|\mathcal{A}| \times |\mathcal{M}|$. Similar to the $ETCF$ matrix, each entry in the $ETKF$ matrix is assumed to be a linear function of the load level on the primary edge of a task, with a positive slope and an intercept that equals zero.

C. Throughput and Latency Requirements

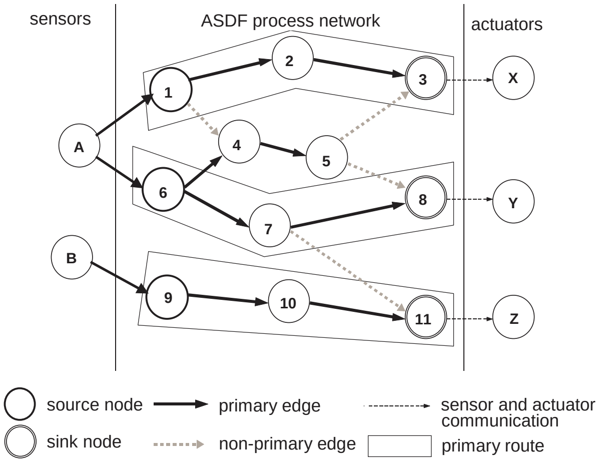

Figure 1. An example sensor-actuator network.

A task a_i, where $a_i \in \mathcal{A}$, may be associated with a throughput requirement, $THREQ[a_i]$. That is, the output data rate of task a_i is required not to be slower than its throughput requirement, $THREQ[a_i]$. Also, a sensor-actuator pair (s_l, t_m), where $s_l \in \mathcal{S}$ and $t_m \in \mathcal{T}$, may be associated with an end-to-end latency requirement, $LREQ[s_l, t_m]$ That is, the time between the sensor s_l sending data out and the actuator t_m receiving a message resulting from the processing of that data cannot exceed $LREQ[s_l, t_m]$. Initially, the throughput and end-to-end latency requirements are specified. We will show an example that uses the concept of primary edge to reasonably assign throughput requirements. Note that the end-to-end latency requirement can be specified for paths that consist of primary edges only.

D. Example

A source node in the ASDF process network represents a task that is connected to a sensor by a primary edge. A sink node represents a task that sends data to actuators. An actuator can receive data from exactly one sink node. This is a natural assumption, which allows an actuator to be controlled by only one task in order to avoid potential conflicts. Each sensor outputs data to the corresponding source node(s) at a fixed rate. This rate can be different for different sensors. The set of source nodes is represented by $\mathcal{A}_{source}$, and the set of sink nodes by $\mathcal{A}_{sink}$. Given a node in $\mathcal{A}$, a route is defined as a set of nodes constituting a path from a source node to the given node through a series of primary edges. ¿From the definition of primary edge, it follows that a route for each node is unique. The route for a sink node is called a primary route (PR). There are exactly $|\mathcal{A}_{sink}|$ primary routes (shown enclosed by boxes in Figure 1). Note that data from a sensor can reach an actuator through a primary route only. Let $\underline{r_k}$ denote a primary route, where $1 \leq k \leq |\mathcal{A}_{sink}|$. The set of all primary routes is represented by $\underline{\mathcal{R}}$. The concept of primary routes is related to the "continuous paths" used in the DeSiDeRaTa project [19].

For each source node $\underline{a_s} \in \mathcal{A}_{source}$, its throughput requirement, $\underline{THREQ}[a_s]$, is set to be the input data rate of this source node. This throughput requirement is imposed on all the nodes along the routes that originate from a_s. Thus, $THREQ[a_i]$, the throughput requirement of task a_i, is equal to $THREQ[a_s]$ if there is a route from a_s to a_i. Let $\underline{LREQ}$ be a vector representing all end-to-end latency requirements such that $LREQ[r_k] = LREQ[s_l, t_m]$, where $r_k \in \mathcal{R}$ and r_k is a primary route between sensor s_l and actuator t_m. Similarly, let $\underline{THREQ}$ be a vector representing all throughput requirements such that $THREQ[a_i]$ is the throughput requirement of task a_i, where $a_i \in \mathcal{A}$.

4.3 Maximum Allowable Increase in Load

As discussed in Section 3, the performance metric of an allocation is given as the highest value of $d_R(R, R_0)$, where R represents a vector of run-time parameters. In our model, each run-time parameter contributes to the load level of each task. Thus, R is represented by the load level vector LL. We assume that there is a run-time parameter, called $\underline{\alpha}$, that governs the increase in the load level of all the tasks. Then, $LL(a_i) = (1 + \alpha) \times LL_{init}(a_i)$. Therefore, α can be used as the norm d_R. Thus, the performance metric of a resource allocation can be defined as the maximum value of α under the condition that there is no throughput or end-to-end latency violation, denoted as $\underline{\alpha_{max}}$. This specific performance metric is called the maximum allowable increase in load ($\underline{MAIL}$).

4.4 Problem Definition

Based on the models described in Sections 4.1 and 4.2, and the performance metric explained in Section 4.3, a formal definition of the resource allocation problem can be stated as follows:

Given:

1. A set of machines $\mathcal{M}$
2. A set of sensors, $\mathcal{S}$, and a set of actuators, $\mathcal{T}$
3. An ASDF process network, $\mathcal{G}(\mathcal{E}, \mathcal{A})$ and a set of all primary edges $\mathcal{E}_P$
4. $ETCF$ and $ETKF$ matrices
5. Vectors LL_{init}, $LREQ$ and $THREQ$

Find:

An allocation of all tasks in $\mathcal{A}$ onto machines in $\mathcal{M}$ that has the highest value of $MAIL$

5 A Mathematical Programming Formulation

5.1 Latencies and Requirements

A mathematical formulation for the problem based on our model is described in this section. An objective function is defined, which leads to an optimal $MAIL$ value under some assumptions. However, the formulation contains non-linear equations. By using a linearization heuristic, the formulation is reduced to a mixed-integer-programming formulation.

Let $\underline{X}$ be a matrix that represents an allocation of tasks onto machines such that

$$X[a_i, m_l] = \begin{cases} 1 & \text{if } m_l \text{ is allocated to } a_i \\ 0 & \text{otherwise} \end{cases}$$

where $a_i \in \mathcal{A}$ and $m_l \in \mathcal{M}$. A task can be allocated to only one machine. Consequently, for any task a_i, there is exactly one $X[a_i, m_l]$ that is equal to 1 for all $m_l \in \mathcal{M}$. Given an allocation, let $\underline{n_l}$ be the total number of tasks that execute on machine m_l, i.e., $n_l = \sum_{a_i \in \mathcal{A}} X[a_i, m_l]$.

In this paper, each of the CPU and communication bandwidth of a machine is assumed to be fairly shared among all the tasks allocated to the machine. Thus, the latency of a task on a machine is estimated as the product of its base latency (without sharing) and the total number of tasks sharing the same machine. Let $\underline{ECL}$ denote the estimated-initial-computation-latency matrix. Each entry in the $ECL[a_i, m_l]$ matrix is the estimated computation latency of task a_i on machine m_l at the initial load level, when the machine is dedicated to this task. ECL can be calculated from the $ETCF$ matrix and the LL_{init} vector. Specifically, $ECL[a_i, m_l] = ETCF[a_i, m_l].a \times LL_{init}(a_i)$. The computation latency of task a_i for the initial load level, denoted $\underline{C(a_i)}$, can be calculated as

$$C(a_i) = \sum_{m_l \in M} \{X[a_i, m_l] \times ECL[a_i, m_l] \times n_l\}$$

Similarly, $\underline{EKL}$ matrix is defined as the estimated-initial-c(K)ommunication-latency matrix. In this paper, a simple communication model is used to simplify the calculation of the communication latency. Because the sensors are part of system hardware and cannot be allocated, the latency of a sensor sending data to the network switch is assumed to be independent of the load level of any task. Similar assumption is made for the latency of an actuator receiving data from the network switch. These latencies can be absorbed in the end-to-end latency requirements of sensor-actuator pairs. The communication latency of a task is modeled in two stages: one for sending data to and the other for receiving data from the network switch. The total communication latency, $K(a_i)$, is

$$K(a_i) = 2 \times \sum_{m_l \in M} \{X[a_i, m_l] \times EKL[a_i, m_l] \times n_l\}$$

The actual end-to-end latency of a PR r_k is

$$L(r_k) \quad = \textstyle\sum_{a_i \in r_k} \{C(a_i) + K(a_i)\}$$

Given a primary route $r_k \in \mathcal{R}$, its end-to-end latency requirement is satisfied when the actual latency does not exceed its requirement. Mathematically,

$$L(r_k) \leq LREQ[r_k]$$

Given a task a_i, its throughput requirement is satisfied when *both* its computation latency and communication latencies do not exceed the reciprocal of its throughput requirement. Mathematically,

$$\max\left(C(a_i), K(a_i)\right) \leq \frac{1}{THREQ[a_i]}$$

5.2 Objective Function

Let $NSC(a_i)$ be the normalized computation slackness of task a_i. It is calculated as:

$$NSC(a_i) = 1 - \frac{C(a_i)}{1/THREQ[a_i]}$$

Similarly, $NSK(a_i)$, the normalized communication slackness of a task a_i is defined as:

$$NSK(a_i) = 1 - \frac{K(a_i)}{1/THREQ[a_i]}$$

For a route $r_k \in \mathcal{R}$, the normalized slackness is

$$NSR(r_k) = 1 - \frac{L(r_k)}{LREQ[r_k]}$$

Conceptually, the normalized slackness represents, as a percentage, the slack or available room for the latency of a task or a route to increase, before the throughput and/or end-to-end latency requirement is violated.

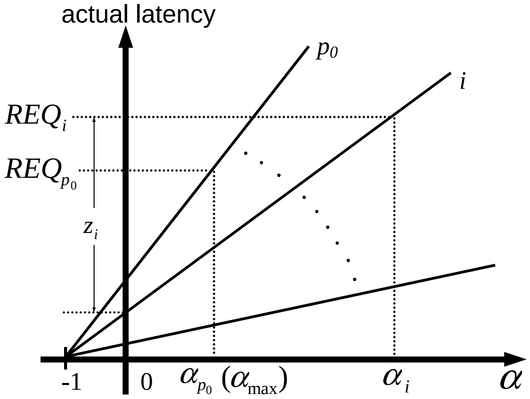

Figure 2. Variation of the actual latency of tasks and primary routes with respect to α.

Note that an allocation satisfies all performance requirements iff the normalized slackness for all tasks and all primary routes is non-negative.

Given an allocation, let

$$\begin{aligned} c_{min} \quad = \min\{ \quad & \min_{r_k \in \mathcal{R}} NSR(r_k), \\ & \min_{a_i \in \mathcal{A}} NSC(a_i), \\ & \min_{a_i \in \mathcal{A}} NSK(a_i)\} \end{aligned}$$

Claim 1: Given a feasible allocation, $c_{min} = \frac{\alpha_{max}}{1+\alpha_{max}}$
Proof: For a given feasible allocation, Figure 2 shows the variation of the actual computation or communication latency of all tasks and the actual end-to-end latency of all primary routes with respect to α. In Figure 2, actual latency refers to the actual computation or communication latency of a task, or the actual end-to-end latency of a primary route. Similarly, REQ_i refers to the constraint on task a_i or primary route r_i ($\frac{1}{THREQ}$ for a task or $LREQ$ for a primary route). Note that when α equals -1, the actual latency of any task or route becomes zero. The slope of the line that corresponds to the actual computation latency of task a_i can be calculated as $\sum_{1 \leq l \leq |\mathcal{M}|}(X[a_i, m_l] \times ETCF[a_i, m_l].a \times n_l)$. By replacing $ETCF$ with $ETKF$ in the previous expression, we get the slope of the line that corresponds to the actual communication latency of task a_i. The slope of the line that corresponds to the actual end-to-end latency of primary route r_k can be calculated as $\sum_{a_i \in r_k}(\sum_{1 \leq l \leq |\mathcal{M}|} X[a_i, m_l] \times (ETCF[a_i, m_l].a + ETKF[a_i, m_l].a) \times n_l)$.

Let p represent the total number of lines in Figure 2. We have $p = 2|\mathcal{A}| + |\mathcal{R}|$. The actual latency for a task or a primary route increases when α increases (i.e., the load level of each task increases). For a task a_i

(primary route r_i), let α_i be the value of α such that the actual latency of a_i (r_i) reaches REQ_i. Given any feasible allocation, we have $\alpha_i \geq 0, 1 \leq i \leq p$.

Based on the definition of α_{max} (the $MAIL$ value of an allocation) in Section 4.3, we have $\alpha_{max} = \min_{1 \leq i \leq p}\{\alpha_i\}$. Thus, there exists some p_0, $1 \leq p_0 \leq p$, such that $\alpha_{max} = \alpha_{p_0}$. Note that,

$$\frac{\alpha_{p_0}}{1+\alpha_{p_0}} = \frac{\min_{1 \leq i \leq p} \alpha_i}{1+\min_{1 \leq i \leq p} \alpha_i} = \min_{1 \leq i \leq p}\{\frac{\alpha_i}{1+\alpha_i}\}, \alpha_i \geq 0$$

From Figure 2, we have $\frac{\alpha_i}{1+\alpha_i} = \frac{z_i}{REQ_i}$. It follows that

$$\begin{aligned}\frac{\alpha_{p_0}}{1+\alpha_{p_0}} &= \min_{1 \leq i \leq p}\{\frac{z_i}{REQ_i}\} \\ &= c_{min}\end{aligned}$$

Thus, we have $c_{min} = \frac{\alpha_{max}}{1+\alpha_{max}}$. □

Claim 2: The allocation that gives the highest value of c_{min} among all allocations will also result in the highest value of α_{max}.

Proof: From Claim 1, we have $c_{min} = \frac{\alpha_{max}}{1+\alpha_{max}}$. Because α_{max} is maximized when $\frac{\alpha_{max}}{1+\alpha_{max}}$ is maximized, maximizing c_{min} maximizes α_{max}. □

Thus, the objective function of our mathematical formulation is to maximize the value of c_{min}.

5.3 MIP Approach

Direct mathematical formulation of the problem contains non-linear equations. Specifically, two variables $X[a_i, m_l]$ and n_l are multiplied together. To linearize the formulation, the number of tasks allocated onto a machine is pre-selected using a linearization heuristic. This is specified in a vector $\underline{N}$, where $N[m_l]$ is the pre-selected number of tasks on machine m_l. A simple heuristic is used to choose this number based on the capability of each machine. An example of a method used to evaluate the capability of a machine in a heterogeneous system can be found in [8].

The problem can now be formulated using mixed-integer-programming, as follows.

Given: $\mathcal{M}, \mathcal{A}, \mathcal{E}_P, ETCF, ETKF, LL_{init}, \mathcal{R}, LREQ, THREQ, N$

Find: X, c

to

Maximize: c

Subject to:

$$\begin{aligned} c &\leq NS(r_k) \quad , \quad \forall r_k \in \mathcal{R} \\ c &\leq NS(C(a_i)) \quad , \quad \forall a_i \in \mathcal{A} \\ c &\leq NS(K(a_i)) \quad , \quad \forall a_i \in \mathcal{A} \\ \sum_{m_l \in \mathcal{M}} X[a_i, m_l] &= 1 \quad , \quad \forall a_i \in \mathcal{A} \\ \sum_{a_i \in \mathcal{A}} X[a_i, m_l] &= N[m_l] \quad , \quad \forall m_l \in \mathcal{M} \end{aligned}$$

At the end of the optimization, the auxiliary c will be equal to c_{min}. Furthermore, X will represent a resource allocation that gives the highest value of c_{min} and thus the highest value of α_{max}.

6 Experimental Results

6.1 Experimental Procedure

A simulator based on the mathematical formulation presented in Section 5 was developed to evaluate the performance of our resource allocation techniques. Given an allocation, the simulator calculates the task execution latencies using the equations presented in Section 5. The performance attributes are then calculated and compared with the corresponding requirements. If all performance requirements are satisfied, the normalized slackness values of all tasks and primary routes are calculated. The $MAIL$ value for this allocation is given by the minimum among all these normalized slackness values.

The simulations were divided into two sets based on the problem size. The first set consisted of small problems that consist of 3-4 machines and 12 tasks. The second set ranged from 10 to 20 machines and 30 to 60 tasks. We also implemented three other approaches: Integer Programming-based (referred to as IP in the following text), Min-Min, and Greedy. IP minimizes the sum of the end-to-end latencies of all PRs, which is similar to the objective function presented in [7]. The Greedy heuristic maps tasks in a random order and each task is mapped onto the machine that gives the shortest computation and communication latency based on the mapping information so far. The Min-Min heuristic is a variation of the algorithm D in [11] that orders tasks using their computation and communication latency on the best machines. In all the three approaches, the computation and communication latencies are calculated while considering multitasking of tasks. The formulation for IP and pseudo codes for the Greedy and Min-Min heuristics can be found in [8]. The performance of MIP was compared with all these three approaches.

For both sets, the $ETCF$ and $ETKF$ matrices were generated to capture the machine and task heterogeneities [1]. Specifically, each matrix was characterized by two parameters: machine heterogeneity and task heterogeneity. Both heterogeneities can be modeled as "high" or "low." Gamma distributions were used to generate the matrices. The four categories of the matrices and the corresponding input parameters are shown in Table 1. In the table, μ quantifies the average value of the slopes in the matrices and V quanti-

heterogeneity $\mathcal{A}-\mathcal{M}$	input parameters			
	μ_{task}	$\mu_{machine}$	V_{task}	$V_{machine}$
Lo-Lo	50	**	0.1	0.1
Lo-Hi	*	30	0.1	0.5
Hi-Lo	50	**	0.5	0.1
Hi-Hi	50	**	0.5	0.5

* is computed for each task as a function of $\mu_{machine}$ and $V_{machine}$
** is computed for each machine as a function of μ_{task} and V_{task}

Table 1. Characteristics of $ETCF$ and $ETKF$ matrices.

fies the heterogeneities. Each element in vector LL_{init} is generated by sampling a uniform distribution from 10 to 100.

For each task, the average values of its computation and communication latencies over all machines were calculated from the ECL and ETK matrices. For each PR, the sum and the maximum value of these average values of all nodes along the route were calculated, denoted as s and m, respectively. Let $\bar{n}_l$ be equal to $|\mathcal{A}|/|\mathcal{M}|$. The end-to-end latency requirement of the PR was then set to $s \times \bar{n}_l \times f$. f is a specified factor that is used to adjust the tightness of the constraints. The throughput requirement of each task along the PR was set to be $1/(m \times \bar{n}_l \times f)$. Due to space limitations, we emphasized only computation intensive applications – the average communication latency for each task is around 1/100 of its average computation latency.

6.2 Results

For small problem instances, an optimal allocation that results in the highest $MAIL$ value was found by enumerating all possible allocations. Because the execution time for enumeration is quite large for problems with more than 4 machines, the number of machines was limited to 4. The number of tasks was fixed at 12 (2 sources, 3 sinks, out-degree ≤ 2). To study the effect of the tightness of constraints, f was set to two different values (1.5 and 2).

The simulation results are shown in Figure 3. The ratios of the $MAIL$ values obtained from various approaches to the optimum $MAIL$ value are presented. Each bar represents the average value of the ratio over 40 instances (samplings of the gamma distributions used to generate the $ETCF$ and $ETKF$ matrices), with a 90% confidence interval and a 20% (or better) precision. The confidence intervals are indicated by lines at the top of each bar. If any approach fails in an instance, that instance is excluded from the calculation of the average ratio and the confidence interval of

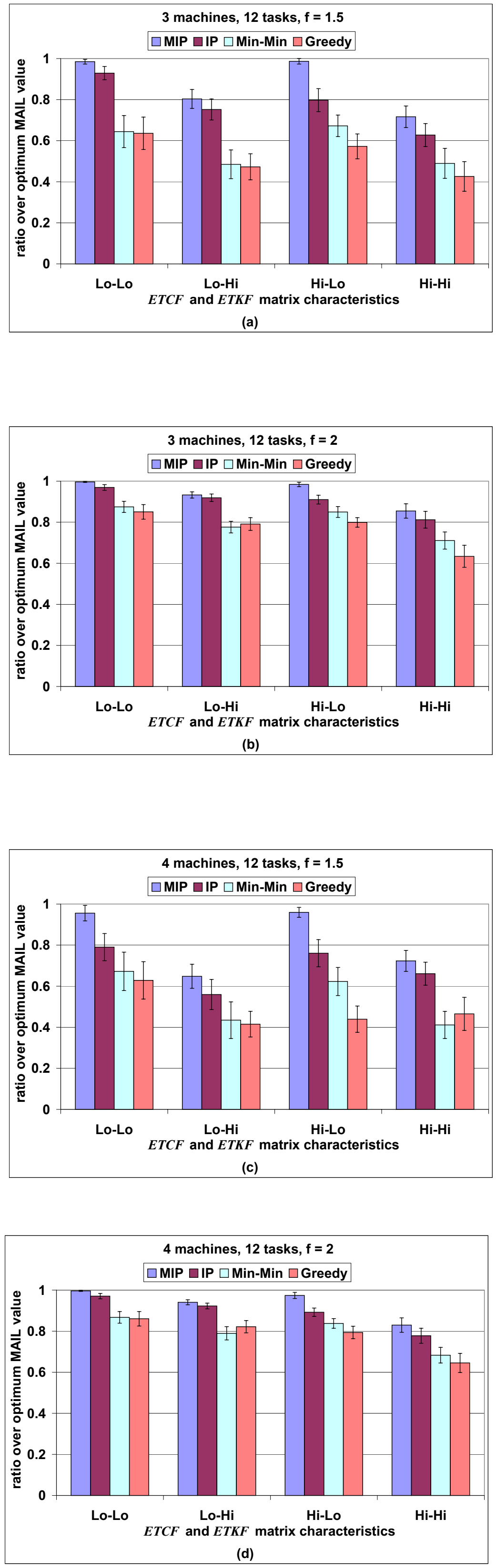

Figure 3. Simulation results for small problems.

the approach. The plots clearly show that on the average, MIP outperforms the other three approaches for all matrix characteristics. In the Hi-Lo environment of Figure 3 (c), the average performance of MIP is 2 times better than that of Greedy. Also, for the Min-Min and Greedy approaches, a 5-25% miss rate (the number of instances that an approach fails normalized with the total number of instances) was observed in the simulations, and this miss rate increases when the value of f decreases. Details can be found in [8].

Simulations for large problems also showed similar trends (see Figure 4). Due to space limitations, results for only a single value of f (1.4) in Lo-Lo and Hi-Hi environments are presented. Because it is impractical to find an optimal allocation by enumeration for large problems, the actual $MAIL$ values for all the four approaches are shown. Each bar represents the average $MAIL$ value over 40 instances with a 90% confidence interval and a 15% (or better) precision. The confidence intervals are indicated by the lines at the top of each bar. If any approach fails in an instance, that instance is excluded from the calculation of the average $MAIL$ value and the confidence interval of the approach. By fixing the number of machines (tasks) and varying the number of tasks (machines), two sets of simulations were performed for large problems.

In the first set, the number of tasks was fixed at 40 (7 sources, 7 sinks, out-degree ≤ 7) while the number of machines ranged from 10 to 20. In the second set, the number of machines was fixed at 12, while the number of tasks ranged from 30 (5 sources, 5 sinks, out-degree ≤ 3) to 60 (10 sources, 10 sinks, out-degree ≤ 3). In all cases, MIP provided the highest average $MAIL$ values. In some cases (e.g., 18 machines and 40 tasks in Figure 4 (a)), MIP showed a performance improvement of 100% (or better) over the other approaches.

Note in Figure 4 (a), the $MAIL$ value does not increase when the number of machines is increased. This is because the value of $\bar{n}_l$ decreases when the number of machines increases (for a fixed number of tasks) and consequently, the throughput and end-to-end latency requirements of the tasks become tighter. Similar explanation holds for the other graphs in Figure 4.

6.3 Performance Comparison

In this section, we compare MIP with the Greedy and Min-Min heuristics defined in Section 6.1 to show the effectiveness of MIP.

Theorem 6.1 *For all $n \geq 4$, where n is the number of tasks to be allocated, there are problem instances for which the Greedy and Min-Min approaches fail to find*

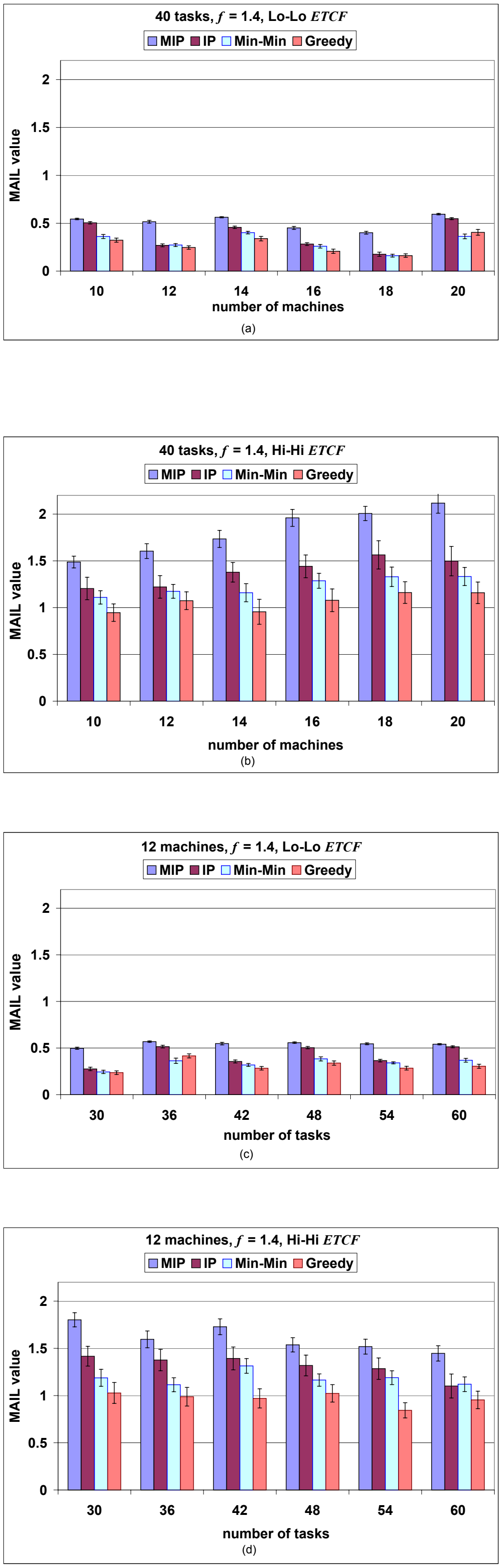

Figure 4. Simulation results for large problems.

a resource allocation that satisfies all performance requirements, while MIP succeeds (in finding a feasible resource allocation).

Detailed proof of Theorem 6.1 can be found in [8]

7 Concluding Remarks

This paper studied a critical problem of efficiently allocating resources in dynamic real-time systems such as embedded hybrid systems. A novel performance metric that considers the effects of run-time parameter variations was proposed. Based on the discussion in Section 3, it would be useful to generalize the formal performance metric by considering several other variations in run-time parameters other than load level. We have made many simplifying assumptions to mathematically formulate the problem. Several interesting variations of the problem can be defined in the future. The performance of the various heuristics depends on f. We plan to study its impact on the performance of the proposed heuristics.

References

[1] S. Ali, H. J. Siegel, M. Maheswaran, D. Hensgen, and S. Ali. Task execution time modeling for heterogeneous computing systems. In *9th Heterogeneous Computing Workshop*, pages 185–199, May 2000.

[2] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. Hard real-time scheduling: The deadline monotonic approach. In *8th IEEE Workshop on Real-Time Operating Systems and Software*, pages 127–132, May 1990.

[3] S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *11th Real-Time Systems Symposium*, pages 182–190, Dec. 1990.

[4] R. Bettati and J. W.-S. Liu. End-to-end scheduling to meet deadlines in distributed systems. In *International Conference on Distributed Computing Systems*, pages 452–459, June 1992.

[5] B. M. Carlson and L. W. Dowdy. Static processor allocation in a soft real-time multiprocessor environment. *IEEE Transactions on Parallel and Distributed Systems*, 5(3):316–320, Mar. 1994.

[6] T. Cuatto, C. Passeronge, L. Lavagno, and et al. A case study in embedded system design: an engine control unit. In *35th Annual Conference on Design automation*, pages 804–807, June 1998.

[7] S. Gertphol, Y. Yu, A. Alhusaini, and V. K. Prasanna. An integer programming approach for static mapping of paths onto heterogeneous real-time systems. In *9th Workshop on Parallel and Distributed Real-Time Systems (WPDRTS) (WPD08.pdf in CDROM for IPDPS 2001)*, Apr. 2001.

[8] S. Gertphol, Y. Yu, S. B. Gundala, V. K. Prasanna, S. Ali, J.-K. Kim, A. A. Maciejewski, and H. J. Siegel. A mixed integer programming based approach to resource allocation for dynamic real-time systems. Technical report, The University of Southern California, 2002, in preparation.

[9] C. J. Hou and K. G. Shin. Allocation of periodic task modules with precedence and deadline constraints in distributed real-time systems. *IEEE Transactions on Computers*, 46(12):1338–1356, Dec. 1997.

[10] C.-W. Hsueh and K.-J. Lin. Scheduling real-time systems with end-to-end timing constraints using the distributed pinwheel model. *IEEE Transactions on Computers*, 50(1):51–66, Jan. 2001.

[11] O. H. Ibarra and C. E. Kim. Heuristic algorithms for scheduling independent tasks on nonidentical processors. *Journal of the ACM*, 24(2):280–289, Apr. 1977.

[12] J. Jonsson and K. G. Shin. A parametrized branch-and-bound strategy for scheduling precedence-constrained tasks on a multiprocessor system. In *International Conference on Parallel Processing(ICPP)*, pages 158–165, Aug. 1997.

[13] Y.-K. Kwok, A. A. Maciejewski, H. J. Siegel, A. Ghafoor, and I. Ahmad. Evaluation of a semi-static approach to mapping dynamic iterative tasks onto heterogeneous computing systems. In *1999 International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN '99)*, pages 204–209, June 1999.

[14] E. Lawler. *Recent Results In The Theory of Machine Scheduling.* Springer Verlag, 1982.

[15] E. A. Lee and T. M. Parks. Dataflow process networks. *Proceedings of the IEEE*, 83(5):773–801, May 1995.

[16] D.-T. Peng, K. G. Shin, and T. F. Abdelzaher. Assignment and scheduling of communicating periodic tasks in distributed real-time systems. *IEEE Transactions on Software Engineering*, 23(12):745–758, Dec. 1997.

[17] K. Ramamritham, J. A. Stankovic, and P.-F. Shiah. Efficient scheduling algorithms for real-time multiprocessor systems. *IEEE Transactions on Parallel and Distributed Systems*, 1(2):184–194, Apr. 1990.

[18] J. Santos, E. Ferro, J. Orozco, and R. Cayssials. A heuristic approach to the multitask-multiprocessor assignment problem using the empty-slots method and rate monotonic scheduling. *Journal of Real-Time Systems*, 13(2):167–199, Sep. 1997.

[19] L. R. Welch, B. Ravindran, B. Shirazi, and C. Bruggeman. Specification and modeling of dynamic, distributed real-time systems. In *19th IEEE Real-Time Systems Symposium (RTSS'98)*, pages 72–81, Dec. 1998.

[20] J. Xu. Multiprocessor scheduling of processes with release times, deadlines, precedence, and exclusion relations. *IEEE Transactions on Software Engineering*, 19(2):139–154, Feb. 1993.