

# MIP Formulation for Robust Resource Allocation in Dynamic Real-Time Systems\*

Sethavidh Gertphol and Viktor K. Prasanna  
University of Southern California  
Department of EE-Systems  
Los Angeles, CA 90089-2562 USA  
{gertphol, prasanna}@usc.edu

## Abstract

*Real-time systems usually operate in an environment that changes continuously. These changes cause the performance of the system to vary during run time. An allocation of resources in this environment must be robust. Using the amount of load variation that the allocation can accommodate as a measure of robustness, we develop a mathematical formulation for the problem of robust resource allocation. Due to the complexity of the models used to represent the problem, the formulation is non-linear. We propose a linearization technique based on variable substitution to reduce the mathematical formulation to a mixed integer programming formulation, called SMIP. Compared with existing techniques, the search space of SMIP is not restricted. Thus, if a feasible allocation exists, SMIP will always produce an optimal allocation.*

**Keywords:** dynamic real-time systems, resource allocation, robust, mixed integer programming, linearization.

## 1 Introduction

Recent developments in dynamic real-time systems such as embedded systems result in widespread deployment of these systems in various application domains [3]. The operating environment, where dynamic real-time systems are deployed, is continuously changing. Changes in the environment often affect performance of these systems. For example, changes in the environment may cause an increase in workload of some application tasks, resulting in degraded performance of those tasks. Another important characteristic of these systems is the real-time requirements imposed upon application tasks. The requirement is often expressed in terms of task deadline, i.e., the amount of time in which

the task must be completed. Violating these real-time requirements might cause catastrophic effects on the system.

When the status of the environment changes causing workloads of some tasks to increase, those tasks may violate their deadline and the current resource allocation becomes invalid. In order to sustain the system in the 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. Since re-allocation is usually needed when tasks lack adequate resources, an overhead is incurred when the system can least afford. To avoid the cost of early re-allocation, resource allocation techniques that consider workload variation during run time have been recently developed [1, 6, 7]. The allocation generated by these techniques will be *robust* with respect to changes in the environment during run time. Task allocation in real-time systems in order to meet certain deadlines is known to be an NP-hard problem [8]. In [6], a mathematical programming formulation is developed to represent the resource allocation problem. The formulation is non-linear, but can be linearized into a mixed integer programming (MIP) formulation by *pre-selecting* the number of tasks running on each machine. The quality of the resulting resource allocation depends largely on these pre-selected numbers. If the pre-selected numbers match the number of tasks running on each machine in an optimal allocation, the resulting allocation will also be optimal. On the other hand, if the numbers are selected poorly, such as where all tasks are allocated to the same machine, the mixed integer programming formulation may not have a feasible solution.

In this paper, a method to linearize the mathematical formulation is proposed. This method is based on a technique of variable substitution and constraint addition in [11]. Because this method does not rely on pre-selection of any number, the resulting allocation is guaranteed to be optimal. The experimental results show that the substitution method

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

generates an allocation that is 1.25 times better than the allocation produced by the pre-selection technique. The execution time of the mixed integer programming formulation, which is linearized by the substitution technique, is acceptable for the problem of initial resource allocation. Throughout this paper, we use the words *resource allocation* and *initial resource allocation* interchangeably.

This paper is organized as follows. A brief discussion about related work is presented in Section 2. Section 3 explains the system and application models used in this paper. A novel performance metric to evaluate a resource allocation for its capability to accommodate changes in the environment is also discussed. A formal mathematical formulation of the resource allocation problem is described in Section 4. The need for linearization and two linearization techniques are explained in Section 5. Experiments and results follow in Section 6. Finally, Section 7 gives conclusion and outlines future work.

## 2 Related Work

Many algorithms and heuristics [9, 12, 13, 14] have been developed to solve the problem of resource allocation in dynamic real-time systems. Most of these resource allocation techniques do not consider the changing environment in which the system operates. Many heuristics allocate resources such that the amount of workload is balanced (evenly distributed) among all resources. These heuristics implicitly assume that future workload variation will be the same for all tasks or resources, which is an oversimplification.

Recently, several resource allocation heuristics that explicitly consider workload variation during run time have been developed [1, 6, 7]. Heuristics presented in [1] are based on a Greedy approach, while in [6] a mixed-integer-programming-based (MIP-based) heuristic is used. Note that the objective function in the mixed integer programming formulation in [6] is similar to an objective value called “system hazard” in [12]. The difference is that the MIP-based heuristic explicitly considers workload variation in mathematical equations constituting the objective function, while the “system hazard” is calculated with no explicit assumption about changing workload.

Mixed integer programming is a subset of a well-researched optimization technique called linear programming (LP). Linear programming has been used extensively in various fields of research, such as Operation Research [11], economics [4], and resource allocations [6, 13]. As the name suggests, variables in mathematical equations constituting the objective function and constraints of LP must be in linear form. However, linear programming formulation of many problems contains non-linear equations. In [6], by pre-selecting the values of one set of variables,

non-linear equations are linearized. On the other hand, in this paper, variable substitution and constraint addition technique derived from [11] is used to linearize the formulation instead.

## 3 Problem Definition

In this section, system and application models are briefly explain. For more detail discussion, please refer to [6].

### 3.1 System Model

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

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.

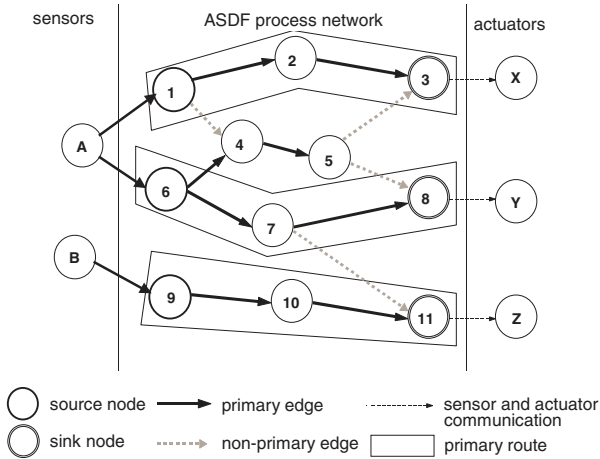
### 3.2 Application Model

The set of processing tasks in real-time systems is modeled using the asynchronous dataflow (ASDF) process network [10]. 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.

#### Run-time parameters

Changes in the environment during run time are captured in parameters called *run-time parameters*. In this paper, *load level* of a processing task is considered to be a parameter that changes during run time. For every processing node that receives data, exactly one incoming edge is set to be a primary edge (dark, solid edges in Figure 1). 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. The *load level* of task  $a_i$ , denoted as  $l_i$ , is a scalar value that represents the data that task  $a_i$  receives through its primary edge.

Let vector  $L$  denote the load level of all tasks, with  $L^{init}$  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



**Figure 1. An example sensor-actuator network**

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.

### Run-time parameter variation

If we allow run-time parameters to vary independently of one another, there will be too many possibilities of the run-time parameter variations. Thus, we assume that there is a run-time parameter, called  $\alpha$ , which governs the changes in the load level of all the tasks. During run time, the load level of a task varies according to the changes in  $\alpha$  and its initial load level. Mathematically,  $l_i = (1 + \alpha) \times l_i^{init}$ , for all  $a_i \in \mathcal{A}$ . Note that when the system starts up,  $\alpha$  is equal to zero, and the current load level of each task is equal to its initial load level.

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  $l_i$  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,  $etcf_{i,l}$ , is assumed to be a linear function of the load level on the primary edge of task  $a_i$ , with a positive slope,  $etcf_{i,l}^a$ , and non-negative intercept,  $etcf_{i,l}^b$ . 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-communicate-function matrix,  $etkf$ , gives a function for each task-machine pair  $(a_i, m_l)$  that maps  $l_i$  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.

### Worst-case resource sharing model

Given the current load level of a task, the task computation and communication latencies can be calculated using the  $etcf$  and  $etkf$  matrices. These latencies are called *base latencies*, or the latencies of a task when resources are not shared. When multiple tasks compete for the same resource, it is assumed that the resource is fairly shared among those tasks. Thus, the *actual latencies* — the latency of a task when a resource is shared — is calculated as the base latency multiplied by the number of tasks competing for that resource. This resource sharing model assumes a worst-case scenario where all tasks attempt to utilize the resource at the same time. Note that the actual latency of a task depends on both its load level, which can vary due to changes in the environment, and on the current resource allocation. This model will be referred to as the *worst-case resource sharing model* throughout the paper.

### Real-time requirements

In this paper, two types of real-time requirements are considered. A task  $a_i$ , where  $a_i \in \mathcal{A}$ , may be associated with a *throughput requirement*,  $THREQ_i$ . That is, the output data rate of task  $a_i$  is required not to be slower than its throughput requirement,  $THREQ_i$ . Also, a primary route  $r_k$  from one sensor to one actuator may be associated with an *end-to-end latency requirement*,  $LREQ_k$ . That is, the time between the sensor sending data out and the actuator receiving a message resulting from the processing of that data cannot exceed  $LREQ_k$ . Let  $LREQ$  and  $THREQ$  be vectors that represent all latency and throughput requirements, respectively. Also let  $R$  be a set of all primary routes.

### 3.3 Performance Metric for Robustness

To avoid early re-allocation, the resource allocation must be *robust* with regard to changing conditions of the environment. One way to measure the robustness of the system is to vary workload and determine the point at which the system violates real-time requirements. As discussed in Section 3.2, a run-time parameter  $\alpha$  governs the variations of all task workload. Thus, the amount that  $\alpha$  can vary can

be used as a performance metric for an allocation. When  $\alpha$  varies, the load level of each task also varies, resulting in the changes in the computation and communication time of each task. However, the computation and communication time of a task must conform with specified real-time requirements. Thus, the allowable variation of  $\alpha$  is limited by real-time requirements. Let  $\alpha_{max}$  be the maximum value of  $\alpha$ , under the condition that there is no deadline violation.  $\alpha_{max}$  depends on real-time requirements and the resource allocation of the system. Thus, we can use  $\alpha_{max}$  of an allocation as a performance metric. This specific performance metric is called *MAIL*, Maximum Allowable Increase in Load level.

## 4 Mathematical Formulation

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  $X$  be a matrix that represents an allocation of tasks onto machines such that

$$x_{i,l} = \begin{cases} 1 & \text{if } m_l \text{ is allocated to } a_i \\ 0 & \text{otherwise} \end{cases}$$

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  $x_{i,l}$  that is equal to 1 for all  $m_l \in \mathcal{M}$ . Given an allocation, let  $n_l$  be the total number of tasks that execute on machine  $m_l$ , i.e.,  $n_l = \sum_{a_i \in \mathcal{A}} x_{i,l}$ .

### 4.1 Latencies and Requirements

The actual computation latency of task  $a_i$  for the initial load level, denoted  $c_i$ , can be calculated as

$$c_i = \sum_{m_l \in \mathcal{M}} \{x_{i,l} \times etcf_{i,l}^a \times l_i \times n_l\}$$

Assume that communication of a task occurs twice for each computation: receiving data from the network, and sending results back after computation. The actual communication latency of task  $a_i$ ,  $k_i$ , is

$$k_i = 2 \times \sum_{m_l \in \mathcal{M}} \{x_{i,l} \times etkf_{i,l}^a \times l_i \times n_l\}$$

Both the actual computation and communication latencies are calculated based on the worst-case resource sharing model. In addition, by assuming that task communication

occurs twice, the actual communication latency takes into account the communication link contention in both direction: from a switch to a machine, and from a machine to a switch. Note that the mathematical formulation for initial resource allocation (Figure 2) is still valid even if other models are used to calculate the actual latency of tasks. However, the formulation may not be reducible to an MIP formulation by using the linearization techniques presented in Section 5.

$p_k$ , the actual end-to-end latency of a PR  $r_k$  is

$$p_k = \sum_{a_i \in r_k} \{c_i + k_i\}$$

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,

$$\max(c_i, k_i) \leq \frac{1}{THREQ_i}$$

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,

$$p_k \leq LREQ_k$$

### 4.2 Normalized Slackness

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

$$nsc_i = 1 - \frac{c_i}{1/THREQ_i}$$

Similarly,  $nsk_i$ , the normalized communication slackness of a task  $a_i$  is defined as:

$$nsk_i = 1 - \frac{k_i}{1/THREQ_i}$$

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

$$nsr_k = 1 - \frac{p_k}{LREQ_k}$$

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

---

**Given:**  $\mathcal{M}, \mathcal{A}, \mathcal{E}_P, \text{etcf}, \text{etkf}, L^{init}, \mathcal{R},$   
 $LREQ, THREQ$

**Find:**  $X, c$

to

**Maximize:**  $c$

**Subject to:**

$$c \leq nsc_i, \quad \forall a_i \in \mathcal{A}$$

$$c \leq nsk_i, \quad \forall a_i \in \mathcal{A}$$

$$c \leq nsr_k, \quad \forall r_k \in \mathcal{R}$$

$$\sum_{m_l \in \mathcal{M}} x_{i,l} = 1, \quad \forall a_i \in \mathcal{A}$$


---

**Figure 2. Mathematical Formulation**

### 4.3 Objective Function

Given an allocation, let

$$c_{min} = \min \left\{ \begin{array}{l} \min_{r_k \in \mathcal{R}} nsr_k, \\ \min_{a_i \in \mathcal{A}} nsc_i, \\ \min_{a_i \in \mathcal{A}} nsk_i \end{array} \right\}$$

It was shown in [6] that the allocation that gives the highest value of  $c_{min}$  among all allocations will also result in the highest value of  $\alpha_{max}$ . Thus, we can use  $c_{min}$  as the objective function for maximization, instead of trying to find  $\alpha_{max}$  directly.

The mathematical formulation of the resource allocation problem is shown in Figure 2. The auxiliary variable  $c$  is a real number, while  $x_{i,l}$ , an entry in  $X$ , is either 0 or 1. The objective is to maximize  $c$ , but  $c$  is limited by the first three constraints corresponding to the normalized slackness of all tasks and primary routes. The last set of constraints enforces a task to be allocated to only one machine. After solving the formulation, the auxiliary variable  $c$  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  $\alpha_{max}$ .

## 5 Linearization Techniques

Direct mathematical formulation of the problem shown in Figure 2 contains non-linear equations. Specifically, to calculate  $nsc_i$ ,  $nsk_i$ , and  $nsr_k$ , two variables  $x_{i,l}$  and  $n_l$  are multiplied together. Recall that  $n_l = \sum_{a_i \in \mathcal{A}} x_{i,l}$ . In [6], the mathematical formulation is linearized by pre-selecting  $n_l$  value for each machine  $m_l$ . In this paper, the variable  $x_{i,l}$  is multiplied into the summation  $\sum_{a_i \in \mathcal{A}} x_{i,l}$ . Then the

- 
1. Select  $n_l$ , for all  $m_l \in \mathcal{M}$ , by using a heuristic.
  2. Solve the following mixed-integer-programming formulation:

**Given:**  $\mathcal{M}, \mathcal{A}, \mathcal{E}_P, \text{etcf}, \text{etkf}, L^{init}, \mathcal{R},$   
 $LREQ, THREQ, N$

**Find:**  $X, c$

to

**Maximize:**  $c$

**Subject to:**

$$c \leq nsr_k, \quad \forall r_k \in \mathcal{R}$$

$$c \leq nsc_i, \quad \forall a_i \in \mathcal{A}$$

$$c \leq nsk_i, \quad \forall a_i \in \mathcal{A}$$

$$\sum_{m_l \in \mathcal{M}} x_{i,l} = 1, \quad \forall a_i \in \mathcal{A}$$

$$\sum_{a_i \in \mathcal{A}} x_{i,l} = n_l, \quad \forall m_l \in \mathcal{M}$$


---

**Figure 3. MIP(\*) Approach**

product of two variables is replaced with one auxiliary variable, with additional constraints introduced. The following two subsections describe both methods.

### Pre-selection

To linearize the formulation, the number of tasks allocated onto a machine is pre-selected using a linearization heuristic. The numbers are specified as a vector  $N$ , where  $n_l$  is the pre-selected number of tasks on machine  $m_l$ . An example of a heuristic used to select  $n_l$  values in a heterogeneous system, called capability-based heuristic (CBH), can be found in [5]. This technique will be referred to as the *pre-selection* technique throughout the paper.

The problem can now be formulated using mixed integer programming, as shown in Figure 3. The main differences between Figure 2 and Figure 3 are: i) the  $N$  vector is given as an input, and ii) additional constraints (the last set) are introduced to force the actual number of tasks on each machine to be equal to the pre-selected number. The approach is called MIP(\*), where \* is substituted by the name of the heuristic used to pre-select the  $n_l$  values. Thus, if CBH is used to pre-select the  $n_l$  values for the mixed integer programming formulation (in step 1 of Figure 3), the entire approach will be called MIP(CBH).

### Variable substitution

In the pre-selection method, the search space of the linearized MIP formulation is limited to the allocations with the actual number of tasks on each machine equal to the pre-selected value. This limitation is mathematically described as the last set of constraints shown in Figure 3. The qual-

ity of the resulting allocation depends largely on the pre-selected number of tasks allocated onto each machine. If these pre-selected numbers are equal to the actual number of tasks on each machine of the optimal solution, MIP(CBH) will also produce an optimal solution. On the other extreme, if these pre-selected numbers are unreasonable (e.g., if all tasks are allocated to the same machine), it is very likely that MIP(CBH) cannot find a feasible allocation at all.

By using the pre-selection method to linearize the MIP formulation, the resulting allocation will not be a “globally” optimal allocation. Feasible allocations with the numbers of tasks on each machine different than the pre-selected numbers will not be considered to be valid solutions, even if they provide better results (higher  $\alpha_{max}$  values). In order to consider these allocations, other linearization techniques that do not restrict the number of tasks allocated to each machine must be used. One way to linearize the formulation is by replacing the product of two variables with one auxiliary variable, together with additional constraints.

Mathematically,

$$x_{i,l} \times x_{j,l} \leq c$$

where  $x_{i,l}$  and  $x_{j,l}$  are binary (0-1) variables,  $i \neq j$ , and  $c$  is some constant, can be replaced with

$$y_{i,j,l} \leq c$$

where  $y_{i,j,l}$  is also a binary variable, with additional constraints

$$\begin{aligned} y_{i,j,l} - x_{i,l} &\leq 0 \\ y_{i,j,l} - x_{j,l} &\leq 0 \\ x_{i,l} + x_{j,l} - y_{i,j,l} &\leq 1 \end{aligned}$$

Note that if  $i = j$ , then no substitution is needed. Also,  $y_{i,j,l}$  and  $y_{j,i,l}$  represent the same product, and thus only one auxiliary variable is used. This technique is a special case of a technique shown in [11].

In mathematical formulation of the resource allocation problem, two variables,  $x_{i,l}$  and  $n_l$ , are multiplied together. However,  $n_l$  can be expanded into  $\sum_{a_i \in \mathcal{A}} x_{i,l}$ . Thus, the linearization technique described earlier can be used. In effect,  $\frac{|\mathcal{A}|^2}{2} \times |\mathcal{M}|$  additional variables and  $3 \times \frac{|\mathcal{A}|^2}{2} \times |\mathcal{M}|$  additional constraints are introduced into the formulation. This specific linearized MIP formulation is not restricted by any pre-selected numbers of tasks on each machine. Thus, solving the formulation will produce a “globally” optimal allocation. This formulation will be referred to as SMIP throughout the paper.

## 6 Experiments and Results

A simulator based on the mathematical formulation presented in Section 4 was developed to evaluate the perfor-

mance of our resource allocation techniques. Given an allocation, the simulator calculates the actual computation and communication latencies of a task using the equations presented in Section 4. If all real-time requirements are satisfied, the normalized slackness values of all tasks and primary routes are calculated, which consequently determine  $c_{min}$ . The  $\alpha_{max}$  value for this allocation can then be calculated using the equation  $\alpha_{max} = \frac{c_{min}}{1-c_{min}}$ .

### 6.1 Problem Generation

The *etcf* and *etkf* matrices were generated to capture the machine and task heterogeneities [2]. Specifically, each matrix was characterized by two parameters: machine heterogeneity and task heterogeneity. Both heterogeneities can be modeled as “Hi” or “Lo.” Gamma distributions were used to generate the matrices. Due to time limitation, the experiment was conducted only in Hi-Hi (high machine and high task heterogeneity) environment. Each element in vector  $L^{init}$  is generated by sampling a uniform distribution of values ranging from 10 to 100.

For each task, the average values of its computation and communication latencies over all machines were calculated from the *etcf* and *etkf* matrices and the  $L^{init}$  vector. 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 Other Heuristics for Comparison

We also implemented two other approaches to solve the resource allocation problem called Min-Min and Greedy. 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 ranks tasks using their computation and communication latencies on the best machines, then allocates the tasks in order. In both approaches, the computation and communication latencies are calculated while considering multitasking of tasks.  $\alpha_{max}$  of the resulting allocations from these two approaches is compared with  $\alpha_{max}$  of the allocations given by MIP(CBH) and SMIP.

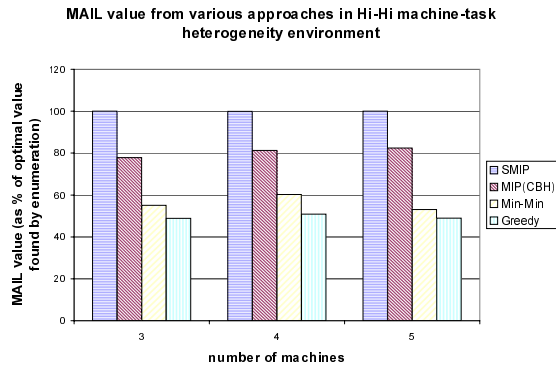


Figure 4. The average MAIL value of allocations from each approach

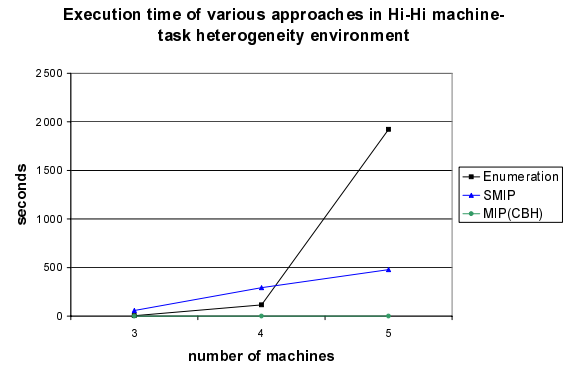


Figure 5. Execution time of each approach

### 6.3 Experimental Procedure

The experiment is divided into 3 sets, each with 40 problem instances. A problem instance in each set is generated with 12 processing tasks (3 sources, 3 sinks, out-degree  $\leq 2$ ). The number of machines varies from 3 to 5 between sets. The  $f$  (tightness of constraint) factor is set to be 1.5 for all problem instances. For each problem instance, an optimal allocation that results in the highest  $\alpha_{max}$  value was found by enumerating all possible allocations. Then, the problem is solved using 4 different approaches: SMIP, MIP(CBH), Greedy, and Min-Min. SMIP is a mixed-integer-programming-based approach, which is linearized using the variable substitution technique. MIP(CBH) is also a mixed-integer-programming-based approach, but is linearized by pre-selecting  $n_i$  values using the CBH heuristic. The MAIL value of the resulting allocation from each approach is then compared with the optimal value found by enumeration. In every problem instance, the execution time of each approach including enumeration, is also recorded.

### 6.4 Results

Figure 4 shows the average MAIL value (as a percentage of the optimal value found by enumeration) of the allocations produced by each approach. As expected, SMIP (MIP-based approach, using substitution technique for linearization) always produces an allocation with the optimal MAIL value. Allocations given by the MIP(CBH) approach has MAIL value around 80% of the optimal value on the average. Min-Min and Greedy generate allocations with the average MAIL value around 55% and 50% of the optimal value, respectively.

Figure 5 shows the average execution time of enumeration, SMIP, and MIP(CBH) on a system with a 400MHz Ultrasparc-II processor and 1GB of main memory. The execution time of enumeration increases very rapidly when

the number of machines in the experiment increase, and is provided as the baseline for comparison. SMIP's execution time also increase with the number of machines, but not as quickly as enumeration's. SMIP takes longer to execute than enumeration in small problems (3 and 4 machines), but when the problem becomes larger, the execution time of enumeration increases rapidly, overtaking the time used by SMIP. For example, to determine an allocation for a system with 5 machines, SMIP takes 500 seconds on the average, compared with 1900 seconds used by enumeration. MIP(CBH) is very fast, using less than 1 seconds in all 3 experiment sets (3, 4, and 5 machines).

From Figure 4, SMIP is obviously the best approach; it always provides allocations with the optimal MAIL value, and is 1.25 times better than allocations from MIP(CBH). The execution time of SMIP is also acceptable for an initial allocation problem. MIP(CBH) runs very fast and produces relatively good results. It is a possible alternative approach when the size of the problem becomes huge or for run-time resource re-allocation.

### 6.5 Discussion

MIP(CBH) can be regarded as a two-phase approach. The first phase allocates resources on a machine-level basis by using the CBH heuristic to pre-select the number of tasks to be allocated onto each machine. The second phase utilizes the mixed integer programming optimization technique to match tasks to machines to maximize  $c_{min}$ , given the pre-selected number from the first phase. From this respect, the actual allocation of resources is determined in the first phase; the MIP formulation in the second phase is used only to optimize the allocation. The resulting allocation will be optimal, only if the pre-selected numbers equal the number of tasks on each machine of an optimal allocation. However, it is possible that the pre-selected numbers will constrain the MIP formulation such that there is no feasible allocation. We can show the following result. Detailed proof can be found in [5].

**Theorem 6.1** For all  $t \geq 2$ , where  $t$  is the number of tasks to be allocated, there are problem instances for which MIP(CBH) fails to find a resource allocation that satisfies all performance requirements, while SMIP succeeds (in finding a feasible allocation).

## 7 Conclusion and Future Work

This paper developed a variable substitution technique to linearize the mixed integer programming (MIP) formulation for robust initial resource allocation in dynamic real-time systems. It was then compared with an existing linearization technique based on pre-selection. Because the substitution technique does not restrict the search space of the MIP formulation, it guarantees to find an optimal allocation. It will also find a feasible allocation, if one exists. The execution time of the MIP formulation linearized by the substitution technique is also acceptable for the problem of initial resource allocation. On the other hand, the pre-selection technique is fast but limits the search space of an MIP formulation to allocations with the number of tasks on each machine equal to the pre-selected numbers. Thus, it may provide sub-optimal results and may fail to find existing feasible allocations.

The short-coming of the pre-selection technique lies in the heuristic used to pre-select the number of tasks on each machine. The CBH heuristic does not consider real-time requirements of the tasks, and thus may generate infeasible allocations. Heuristics that consider these requirements will be developed. Also, the importance of the pre-selected number on the performance of the resulting allocation has been shown in this research. Heuristics that pre-select the optimal or close to optimal number of tasks on each machine will also be considered.

## References

- [1] S. Ali, J.-K. Kim, H. J. Siegel, A. A. Maciejewski, Y. Yu, S. B. Gundala, S. Gertphol, and V. K. Prasanna. Greedy heuristics for resource allocation in dynamic distributed real-time heterogeneous computing systems. In *Proceeding of the International Conference on Parallel and Distributed Processing Techniques and Applications*, June 2002.
- [2] 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.
- [3] T. Cuatto, C. Passeronge, L. Lavagno, A. Jureska, A. Damiano, C. Sansoe, and A.L. Sangiovanni-Vincentelli. A case study in embedded system design: an engine control unit. In *35th Annual Conference on Design automation*, pages 804–807, June 1998.
- [4] R. Dorfman, P. A. Samuelson, and R. M. Solow. *Linear Programming and Economic Analysis*. McGraw-Hill, 1958.
- [5] S. Gertphol and V. K. Prasanna. Robust resource allocation techniques in dynamic real-time systems. Technical report, Department of EE-Systems, University of Southern California, 2003. in preparation.
- [6] 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 for resource allocation in dynamic real-time systems. Technical report, The University of Southern California, 2002, in preparation.
- [7] T. Hegazy and B. Ravindran. Using application benefit for proactive resource allocation in asynchronous real-time distributed systems. *IEEE Transactions on Computers*, 51(8):945 – 962, August 2002.
- [8] 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.
- [9] S. Kartik and C. Siva Ram Murthy. Task allocation algorithms for maximizing reliability of distributed computing systems. *IEEE Transactions on Computers*, 46(6):719 – 724, June 1997.
- [10] E. A. Lee and T. M. Parks. Dataflow process networks. *Proceedings of the IEEE*, 83(5):773–801, May 1995.
- [11] G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd, editors. *Optimization: Handbooks in Operations Research and Management Science*, volume 1. North-Holland, 1989.
- [12] 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.
- [13] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. Practical solutions for qos-based resource allocation problems. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 296–306, December 1998.
- [14] 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.