Submitted to Distributed Computing Systems Symposium, Oct 1994

Analysis of Resource Lower Bounds in Real-Time Applications

Raed Alqadi Parameswaran Ramanathan

Department of Electrical and Computer Engineering
University of Wisconsin-Madison
Madison, WI 53706–1691
parmesh@ece.wisc.edu, (608) 263-0557

ABSTRACT

Tasks in a real-time application usually have several stringent timing, resource, and communication requirements. Designing a distributed computing system which can meet all these requirements is a challenging problem. In this paper, we alleviate this problem by proposing a technique to determine a lower bound on the number of processors and resources required to meet the constraints of the application. We also extend the technique to estimate the cost of a system which meets all the application constraints.

The proposed technique deals with most constraints found in real-time applications including deadlines, release times, resource requirements, precedence relationships, and non-zero communication times. It also derives these bounds for two different models of distributed systems; the shared model in which the resources are shared by all processors and the dedicated model in which each processor has a set of resources dedicated to it.

The results in this paper can be used to reduce the search times for computer-aided synthesis of distributed real-time systems. They can also serve as a baseline for evaluating scheduling algorithms for real-time applications.

Index Terms: Real-time applications, Lower bound analysis, Resource requirements, Distributed computing system.

The work reported here is supported in part by the National Science Foundation grant MIP-9213716.

1 Introduction

Unlike non-real-time applications, tasks in a *real-time* application have deadline constraints by which they must complete their computation. Failure to complete a computation within its deadline may lead to a catastrophe. For example, in a surface ship radar application [8], an incoming missile must be identified within 0.2 seconds of its detection. If necessary, intercept missiles must be engaged within 5 seconds after detection, and launched within 0.5 seconds within engagement. Failure to meet these timing constraints may result in severe destruction of life and property. Other such applications include flight-control systems, life support systems, nuclear power plants, and industrial process-control systems.

Due to the severity of the consequences, a distributed computing system is often dedicated to the tasks in a real-time application. This system must be carefully designed to ensure that all the constraints of the application are satisfied. However, designing such a computer system is a very difficult problem. Several ongoing research projects are developing algorithms/heuristics to facilitate the search and evaluation of various design alternatives [2,7,4]. These heuristics often require an estimate of the number and the type of processors and resources necessary to meet the constraints of the application. In this paper, we propose a method to compute a lower bound on the number of processors and resources of each type required by the application. These lower bounds can also serve as a baseline for evaluating the effectiveness of various scheduling and synthesis heuristics.

Prior work in lower bound analysis has focussed mainly on non-real-time applications. A typical concern in non-real-time applications is to determine a lower bound on the time required to complete a given application on a specified number of processors and resources. For instance, in [6,9–11], lower bound analysis is used to characterize speedups that can be achieved in non-real-time parallel programs. This concern is usually not relevant to real-time applications, where the objective is to complete the tasks in the application within their respective deadlines; it is not necessary to complete the application in as short a time as possible if all the deadline constraints are satisfied. Therefore, in real-time applications, the problem is to determine the number of processors and resources required to meet the constraints of the application.

Pioneering work on determining the number of processors required for a given application was done by Fernandez and Bussell [3]. They consider applications in which the task have integer execution times, precedence relationships, but, zero communication time with other tasks. Furthermore, all tasks are assumed to be non-preemptive and they all require the

same type for processor for execution. For this class of applications, Fernandez and Bussell proposed a method for computing a lower bound on the number of processors required to complete the application within its critical time.

More recently, Al-Mohummed [1] extended the algorithm by Fernandez and Bussell to applications in which the communication requirements between the tasks is non-zero. This extension is significant to our work because it deals with one more constraint commonly found in real-time applications. However, neither Al-Mohummed's algorithm nor any other existing algorithm deals with some of the other constraints crucial to real-time applications. In particular, none of the algorithms deal with deadline constraints. They also do not consider applications in which the tasks can differ on the type of processor on which they can execute. Moreover, they do not consider applications in which the tasks require resources other than processors during execution.

In contrast, in this paper, we consider applications in which the tasks have deadline constraints, release-time constraints, resource requirements, precedence relationships, and non-zero communication time between a task and its predecessor. Furthermore, the tasks can differ on the type of processors on which they can execute and they can be either preemptive or non-preemptive. For such applications, we propose a technique to determine a lower bound on the number of processors and resources of each type. We also evaluate a lower bound on the cost of a system which meets all the application constraints.

The rest of the paper is organized as follows. In Section 2 the assumed model for the application and the distributed system are presented. A brief overview of the proposed lower bound analysis technique is presented in Section 3. The details of the various steps in the lower bound analysis are described in Section 4, 5, 6, and 7. An example to illustrate the proposed technique is presented in Section 8. The paper concludes with Section 9.

2 Problem Formulation

Given an application and a model for the distributed system, the problem is to determine a lower bound on the number of resources of each type required by the application. Using these bounds, an additional problem is to determine a lower bound on the cost of a distributed system which can meet the constraints of the application. In the rest of this section, we present the assumed models for the application and the distributed system.

2.1 Application model

The real-time applications considered in this paper can be modeled as a directed acyclic graph in which the vertices represent the tasks and the edges represent the precedence constraints. The vertices can be further annotated to represent other constraints of the corresponding task. More specifically, each vertex can be annotated with the computation time, the release time, the deadline, and the resource requirements of the corresponding task. The type of processor on which the task executes and whether or not the task is preemptable can also be specified for each vertex. The directed edges can be annotated with the amount of information (i.e., the size of the message) that is sent between the corresponding pair of tasks.

More formally, we use the following notation to represent the various parameters related to an application.

$\mathcal{S}$	The set of tasks in the application
Pred_i	The set of immediate predecessors of task i
Succ_i	The set of immediate successors of task i
C_i	The computation time of task i
rel_i	The release time of task i
D_i	The deadline of task i
ϕ_i	The type of processor on which task i executes
R_i	The set of resources required by task i
m_{ji}	The size of the message sent from task $j \in \text{Pred}_i$ to task i
RES	The set of all resources required by the application, i.e., $\text{RES} = \bigcup_{i \in \mathcal{S}} (\text{R}_i \cup \phi_i)$.

2.2 Distributed system model

In this paper, we consider two different architectures of distributed systems. The architectures differ in the way in which the processors access resources in the system.

Dedicated model. In the dedicated model, the distributed system is assumed to be comprised of nodes of several distinct node types, where each node type is characterized by a processor of a given type and a set of resources dedicated to the processor. The set of distinct node types from which the distributed system is to be constructed is an input to the lower bound analysis. A task can execute on a node which has all the resources needed

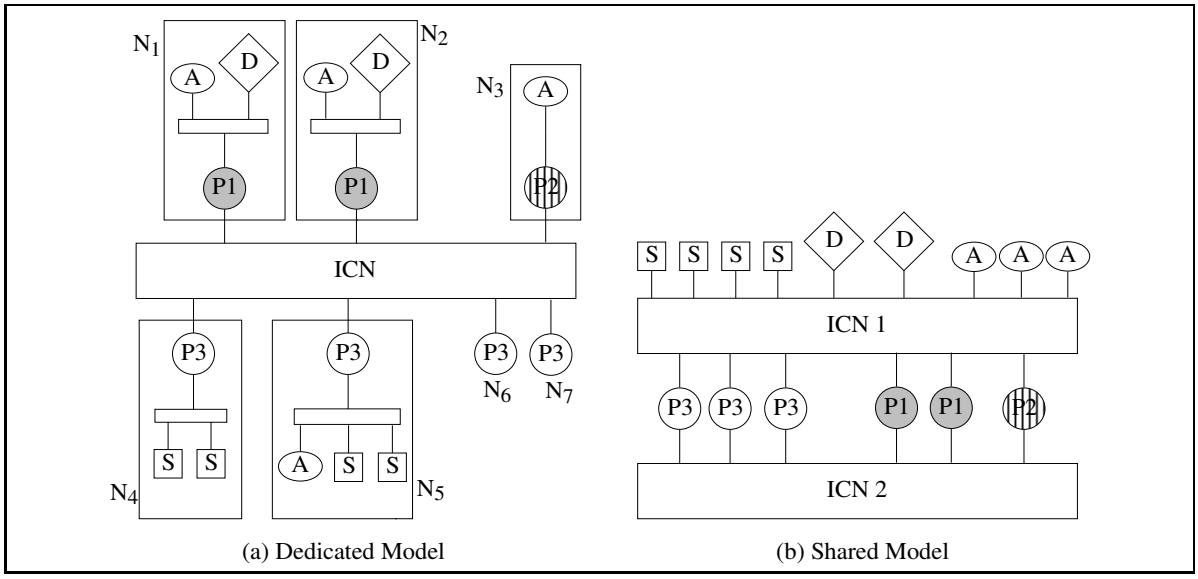

Figure 1: Example of distributed system architectures.

by the task and has a processor that is appropriate for the task. Tasks assigned to two different nodes communicate by sending messages on the interconnection network between the nodes. Tasks assigned to the same node do not send any message on the network to communicate with each other. The communication between a task and its resources is considered to be internal to the node. We assume that for each task there is at least one node (of the appropriate type) which has all the resources it needs.

For example, Figure 1(a) shows an example of a dedicated model for the distributed system. In this example system, there are seven nodes with five distinct types of nodes. Node N1 is comprised a processor of type P1 and a set of resources A and D. Likewise, Node N3 is comprised of a processor of type P2 and resource A. Nodes N1 and N2 are two units of the same type. The interconnection network between the nodes is labeled ICN.

Shared model. In the shared model, all resources in the system are accessible to all processors. However, a resource can be accessed by only one processor at any given time. A task can be assigned to any processor of appropriate type since the processor can access the resources needed by the task. Tasks assigned to two different processors communicate by sending a message on the interconnection network between the processors. Communication between a processor and its resources is assumed to take place on a separate interconnection network between the processors and the resources.

Figure 1(b) shows an example of a shared model distributed system. In this system, there are six processors which share the four units of resource S, two units of resource D, and three units of resource A through an interconnection network labeled ICN1. The communication between the processors takes place on the interconnection labeled ICN2. The differences in the shading of the circles indicate that the corresponding processors are of different types.

Cost model. In the shared model, there is a cost associated with each type of resource and processor that can be used in the system. The total cost of a distributed system is the sum of the costs of the individual costs in the system. In the dedicated model, we associate a cost with each distinct node type. The total cost of a dedicated model distributed system is assumed to the sum of the costs of the nodes in the system. In the shared as well as the dedicated models, the cost of the interconnection networks are ignored. This is reasonable because the cost of the resources like sensors and actuators often dominate the cost of the total system.

The notations used in this paper related to the distributed system model are as follows.

Λ	The set of all node types available for use in the dedicated model
λ_n	The set of resources in a node of type n
CostR(r)	The cost of a resource or a processor of type r
CostN(n)	The total cost of a node of type n.

3 Overview of Lower Bound Analysis

Recall that, the problem is to determine a lower bound, LB_r, on the number of units of each processor/resource $r \in$ RES. An additional problem is to determine a lower bound on the cost of a system which meets the application constraints. The proposed approach for this lower bound analysis involves the following four steps.

1. Compute the earliest start time (EST) and the latest completion time (LCT) of each task in the application.
2. Partition the application tasks into a sequence of smaller subsets such that each subset can be treated independently with respect to lower bound analysis.
3. Compute a lower bound on the number of units of each processor/resource required by the application.

4. Compute a lower bound on the cost of a distributed computing system which can meet the constraints of the application.

Our algorithm for the first step is based on the ideas in [1]. However, unlike the algorithm in [1], our algorithm handles constraints such as resource requirements, release times, and deadlines. The proposed algorithm for the second step is similar to the scheme by Jain and Rajaraman [5]. However, unlike their scheme, the proposed algorithm deals with tasks with arbitrary execution times and non-zero communication times. Our analysis for the third step can deal with both preemptive and non-preemptive tasks. In contrast, the analysis in [1,3] are only for non-preemptive tasks. Finally, lower bound analysis of system costs have not been addressed in literature. This analysis is particularly useful in speeding up the search process in automated synthesis of distributed real-time systems.

In the following sections, we discuss each of these steps in more detail.

4 Evaluation of EST and LCT

The evaluation of LCT and EST of the tasks in the application is simple if there are no communication requirements between the tasks. Otherwise, the evaluation is substantially more involved. One possible approach for dealing with the communication requirements is proposed in [1]. In this paper, we modify that approach to deal with the additional constraints.

Definition 1: In the shared model, a set of tasks $\{i_1, i_2, \ldots, i_k\}$ are said to be *mergeable* if they can all execute on a processor of the same type, i.e.,

$$\phi_{i_1} = \phi_{i_2} \ldots = \phi_{i_k}.$$

Definition 2: In the dedicated model, a set of tasks $\{i_1, i_2, \ldots, i_k\}$ are said to be *mergeable* if there exists a node type which can execute all the tasks in the set, i.e.,

(i) $\phi_{i_1} = \phi_{i_2} \ldots = \phi_{i_k}$, and

(ii) there exists a node type n with a processor of type ϕ_{i_1} and with resources $\lambda_n \supseteq \cup_{l=1}^{k} R_{i_l}$.

Basically, a set of tasks are said to be mergeable if the tasks can be executed on the same processor/node in the given system model. This notion is not considered in [1] because, in

[1], all tasks are assumed to be executable on the same type of processor and the resource constraints are not considered.

4.1 Evaluation of LCT

Consider the evaluation of L_i, the LCT of a task $i \in \mathcal{S}$. If i has no immediate successors, then L_i is equal to its deadline. Otherwise, L_i is recursively computed as follows.

Assume that the LCT of the immediate successors of i has already been computed. If i and one of its immediate successors j are assigned to different processors/nodes, then the message from i must reach j by time $L_j - C_j$ so that j can complete its execution by time L_j. Therefore, i must complete its execution and send a message to j by time $L_j - C_j - m_{ij}$. We refer to $L_j - C_j - m_{ij}$ as the *latest message send* time of i with respect to j and denote it by lms_j.

Now let $A \subseteq \text{Succ}_i$ be a set of mergeable successors of i such that $A \cup \{i\}$ is also mergeable. Consider the effect of assigning the tasks in A to the same processor/node as i and the remaining immediate successors to processors/nodes other than that of i. Then, the tasks in A must be scheduled sequentially after i on the same processor/node as i. Let $\text{lst}(A)$ denote the latest start time for a schedule of the tasks in A on a single processor subject to their LCT constraints. Task i must clearly complete by time $\text{lst}(A)$. By definition, task i must also complete before its deadline D_i. Finally, task i must complete sufficiently early to send messages to tasks $j \in (\text{Succ}_i - A)$ by time lms_j. Combining these observations, the latest completion time for task i given that it is merged with tasks in A is

$$\text{lct}_i(A) = \min\left\{D_i, \quad \min_{\text{Succ}_i - A} \text{lms}_j, \quad \text{lst}(A)\right\}. \tag{4.1}$$

In this equation, the only term we have not specified is $\text{lst}(A)$. This term can be computed as follows. Without loss of generality, let $A \equiv \{i_1, i_2, \ldots i_k\}$ be such that the LCT of i_1 is greater than or equal to that of i_2, which in turn, is greater than or equal to that of i_3, and so on. Generate a schedule by sequentially considering the task in the order i_1, i_2, ..., i_k. Consider the scheduling of task i_j. At this time, tasks i_1, ..., i_{j-1} have already been scheduled. Let e_{j-1} be the start time of i_{j-1} in this schedule. Schedule i_j such that it completes at $\min\{e_{j-1}, L_{i_j}\}$, where L_{i_j} is the latest completion time of i_j. The $\text{lst}(A)$ is then the start time of the task i_k in the resulting schedule.

Equation 4.1 specifies the latest completion time of i if it were to be merged with the

1. If $\text{Succ}_i = \emptyset$, set $L_i = D_i$ and **exit**.
2. Initialize
   $L_i^0 = \min\{D_i, \min_{\{j \in \text{Succ}_i - MS_i\}} \text{lms}_j\}, \quad G_i^0 = \emptyset, \quad k = 1$
3. **while** $(k \leq |MS_i|)$ {
   (a) $T^k = \arg\min_{\{j \in MS - G_i^{k-1}\}} \text{lms}_j$
   (b) **if** $(G_i^{k-1} \cup \{T^k\}$ is mergeable), $G_i^k = G_i^{k-1} \cup \{T^k\}$
   else goto step 4.
   (c) $L_i^k = \min\left\{L_i^0, \min_{j \in MS_i - G_i^k} \text{lms}_j, \text{lst}(G^k)\right\}$
   (d) **if** $(L_i^k \leq L_i^{k-1})$ goto step 4
   else $k = k + 1$
   }
4. $L_i = L_i^{k-1}$, $G_i = G_i^{k-1}$.

Figure 2: Proposed algorithm for evaluating LCT of task i.

tasks in a set A. The problem, therefore, is to find a set A^* which results in the largest latest completion time for i, i.e.,

$$\text{lct}_i(A^*) = \max_{A \cup \{i\} \text{mergeable}} \text{lct}_i(A) = L_i.$$

An efficient method for finding such an A^* is shown in Figure 2.

The basic idea of the algorithm can be explained as follows. If a immediate successor j is merged with i, then i need not send a message to j, thus, saving the time required to send the message. As a result, the completion time of i can be potentially delayed if i and j are merged. On the other hand, if i and j are merged, then they must be sequentially scheduled on the same processor/node. This tends to push the completion of i to an earlier time. Depending on the situation, one of these two factors dominates over the other. If the first factor dominates, then it is beneficial to merge i and j. Otherwise, it is better to assign i and j to two different processors/nodes. The algorithm basically evaluates this tradeoff in a systematic fashion. It terminates as soon as it determines that the completion time of i cannot be increased any further by merging more tasks with i.

More specifically, let $MS_i = \{j : j \in \text{Succ}_i \text{ and } j \text{ mergeable with } i\}$ be the set of immediate successors which are individually mergeable with i. The algorithm considers merging of tasks in MS_i with i in the increasing order of their latest message send time. Let us suppose that $k-1$ tasks have already been merged with i; let G_i^{k-1} denote the set of these $k-1$ tasks. We consider the merging of the k^{th} task, denoted by T^k. If the tasks in $G_i^{k-1} \cup \{T^k\}$ are not mergeable, then the merging process terminates with $A^* = G_i^{k-1}$ because the LCT of i cannot be increased any beyond lms_{T^k}. Otherwise, we evaluate the effect of merging T^k. We compute the latest completion time of i assuming that T^k is also merged with G_i^{k-1}, i.e., we use Equation 4.1 with $A \equiv G_i^{k-1} \cup \{T^k\}$. If the resulting latest completion time is less than the latest completion time before merging T^k, then the process stops with $A^* = G_i^{k-1}$. Otherwise, T^k is merged with G_i^{k-1} and the next task in order is considered.

Theorem 1 below proves the correctness of the algorithm in Figure 2. That is, it shows that the value returned by the algorithm is an upper bound on the latest completion time of the corresponding task.

Theorem 1: For each task $i \in \mathcal{S}$, the value returned by the algorithm in Figure 2 is an upper bound on the latest completion time of i.

Proof: Consider a task i. Let L be latest completion time of i returned by the algorithm. Also, let G be the corresponding set of successors of i which are merged with i in the algorithm. Let $\bar{G} = \text{Succ}_i - G$. We now show that if task i completes later than L then at least one application task will miss its deadline.

From the algorithm, we know that

$$L = \min\{D_i, \text{lst}(G), \min_{j \in \bar{G}} \text{lms}_j\} \text{ and} \tag{4.2}$$

$$\text{lms}_j \leq L \quad \forall j \in G \tag{4.3}$$

Consider a $G' \neq G$. Let $\bar{G}' = \text{Succ}_i - G'$. Let L' be the latest completion time of i if the tasks in G' are merged with i. Then,

$$L' = \min\{D_i, \text{lst}(G'), \min_{j \in \bar{G}'} \text{lms}_j\}. \tag{4.4}$$

We must prove that $L' \leq L$.

Case 1: $G \not\subset G'$

Consider a task $T \in \bar{G}' \cap G$. Hence, from Equations 4.4 and 4.3, $L' \leq \text{lms}_T \leq L$.

Case 2: $G \subseteq G'$

Let $T = \arg\min_{j \in \bar{G}} \text{lms}_j$, i.e., T is the value of j which corresponds to the minimum lms_j. Now consider two cases.

Case 2a: $T \in G'$.

Since $T \in G'$, T is mergeable with G. Therefore, it follows from the algorithm that $\text{lst}(G \cup \{T\}) \leq L$. The result then follows because

$$\begin{aligned} L' &\leq \text{lst}(G') \quad \text{(from Equation 4.4)} \\ &\leq \text{lst}(G \cup \{T\}) \quad \text{(since } G \cup \{T\} \subseteq G') \\ &\leq L. \end{aligned}$$

Case 2b: $T \notin G'$.

From Equation 4.2 and the definition of T, $L = \min\{D_i, \text{lst}(G), \text{lms}_T\}$. Similarly, from Equation 4.4 and the definition of T, $L' = \min\{D_i, \text{lst}(G'), \text{lms}_T\}$. Since $G \subseteq G'$, $\text{lst}(G') \leq \text{lst}(G)$. From these three observations, we can conclude that $L' \leq L$.

From Cases 1, 2a, 2b, we can conclude that the subset G identified by the algorithm results in the largest latest completion time for i. Hence, the algorithm evaluates an upper bound on the latest completion time of each task in the application. □

4.2 Evaluation of EST

Consider the evaluation of E_i, the EST of a task $i \in \mathcal{S}$. If i has no immediate predecessors, then E_i is equal to its release time. Otherwise, E_i is recursively computed as follows.

Assume that the EST of the immediate predecessors of i has already been computed. If i and one of its immediate predecessors j are assigned to different processors/nodes, then the message from j will reach i by time $E_j + C_j + m_{ji}$. We refer to $E_j + C_j + m_{ji}$ as the *earliest message receive* time of i with respect to j and denote it by emr_j.

Now let $A \subseteq \text{Pred}_i$ be a set of mergeable predecessors of i such that $A \cup \{i\}$ is also mergeable. Consider the effect of assigning the tasks in A to the same processor/node as i and the remaining immediate predecessors to processors/nodes other than that of i. Then, the tasks in A must be scheduled sequentially before i on the same processor/node as i.

1. If $\text{Pred}_i = \emptyset$, set $E_i = \text{rel}_i$ and **exit**.
2. Initialize $E_i^0 = \max\{\text{rel}_i, \max_{\{j \in \text{Pred}_i - MP_i\}} \text{emr}_j\}$, $\quad M_i^0 = \emptyset$, $\quad k = 1$
3. **while** $(k \leq |MP_i|)$ {
   (a) $T^k = \arg\max_{\{j \in MP - M_i^{k-1}\}} \text{emr}_j$
   (b) **if** $(M_i^{k-1} \cup \{T^k\}$ is mergeable), $\quad M_i^k = M_i^{k-1} \cup \{T^k\}$
   else goto step 4.
   (c) $E_i^k = \max\left\{\text{rel}_i^0,\ \max_{j \in MP_i - M_i^k} \text{emr}_j,\ \text{ect}(M^k)\right\}$
   (d) **if** $(E_i^k \geq E_i^{k-1})$ goto step 4
   else $k = k + 1$
   }
4. $E_i = E_i^{k-1}$, $M_i = M_i^{k-1}$.

Figure 3: Proposed algorithm for evaluating EST of task i.

Let $\text{ect}(A)$ denote the earliest completion time for a schedule of the tasks in A on a single processor subject to their EST constraints. Task i can potentially start by time $\text{ect}(A)$. By definition, task i cannot start before its release time rel_i. Finally, task i cannot start before it receives messages from tasks $j \in (\text{Pred}_i - A)$, i.e., before time emr_j. Combining these observations, the earliest start time for task i given that it is merged with tasks in A is

$$\text{est}_i(A) = \max\left\{\text{rel}_i,\ \max_{\text{Pred}_i - A} \text{emr}_j,\ \text{ect}(A)\right\}. \tag{4.5}$$

In this equation, the only term we have not specified is $\text{ect}(A)$. This term can be computed as follows. Without loss of generality, let $A \equiv \{i_1, i_2, \ldots i_k\}$ be such that the EST of i_1 is less than or equal to that of i_2, which in turn, is less than or equal to that of i_3, and so on. Generate a schedule by sequentially considering the task in the order i_1, i_2, ..., i_k. Consider the scheduling of task i_j. At this time, tasks i_1, ..., i_{j-1} have already been scheduled. Let c_{j-1} be the completion time of i_{j-1} in this schedule. Schedule i_j such that it starts at $\max\{c_{j-1}, E_{i_j}\}$, where E_{i_j} is the earliest start time of i_j. The $\text{ect}(A)$ is then the completion time of the task i_k in the resulting schedule.

Equation 4.5 specifies the earliest start time of i if it were to be merged with the tasks

in a set A. The problem, therefore, is to find a set A^* which results in the smallest earliest start time for i, i.e.,

$$\text{est}_i(A^*) = \min_{A \cup \{i\} \text{mergeable}} \text{est}_i(A) = E_i.$$

An efficient method for finding such an A^* is shown in Figure 3.

The basic idea of the algorithm can be explained as follows. If a immediate predecessor j is merged with i, then i need not receive a message to j, thus, saving the time required to send the message. As a result, the start time of i can be potentially reduced if i and j are merged. On the other hand, if i and j are merged, then they must be sequentially scheduled on the same processor/node. This tends to push the start of i to a later time. Depending on the situation, one of these two factors dominates over the other. If the first factor dominates, then it is beneficial to merge i and j. Otherwise, it is better to assign i and j to two different processors/nodes. The algorithm basically evaluates this tradeoff in a systematic fashion. It terminates as soon as it determines that the start time of i cannot be decreased any further by merging more tasks with i.

More specifically, let $MP_i = \{j : j \in \text{Pred}_i \text{ and } j \text{ mergeable with } i\}$ be the set of immediate predecessors which are individually mergeable with i. The algorithm considers merging of tasks in MP_i with i in the decreasing order of their earliest message receive times. Let us suppose that $k-1$ tasks have already been merged with i; let M_i^{k-1} denote the set of these $k-1$ tasks. We consider the merging of the k^{th} task, denoted by T^k. If the tasks in $M_i^{k-1} \cup \{T^k\}$ are not mergeable, then the merging process terminates with $A^* = M_i^{k-1}$. Otherwise, we evaluate the effect of merging T^k. We compute the earliest start time of i assuming that T^k is also merged with M_i^{k-1}, i.e., we use Equation 4.5 with $A \equiv M_i^{k-1} \cup \{T^k\}$. If the resulting earliest start time is greater than the earliest start time before merging T^k, then the process stops with $A^* = M_i^{k-1}$. Otherwise, T^k is merged with M_i^{k-1} and the next task in order is considered.

Theorem 2 below proves the correctness of the algorithm in Figure 3. That is, it shows that the value returned by the algorithm is a lower bound on the earliest start time of the corresponding task. The proof of the theorem is similar to the proof of Theorem 1 and is included in the appendix.

Theorem 2: For each task $i \in \mathcal{S}$ the value returned by the algorithm in Figure 3 is a lower bound on the earliest start time of i.

1. Rename tasks in ST_r from 1 to $|ST_r|$ in the increasing order of their ESTs. That is, $i \in ST_r$, $j \in ST_r$, and $i < j$ implies $E_i < E_j$.

2. Initialize $k = 1$, and $P_{r1} = \{1\}$.

3. **For** $i = 2$ **to** $|ST_r|$

   If $E_i < \max_{j \in P_{rk}} L_j$ **then**
   $P_{rk} = P_{rk} \cup \{i\}$
   Else
   $k = k + 1$; $P_{rk} = \emptyset$
   Endif

   Endfor

Figure 4: Proposed procedure for partitioning the application tasks.

5 Partitioning of Application Tasks

To reduce the complexity of the lower bound analysis (cf. Section 6), it is beneficial to partition the application tasks into smaller subsets which can be treated independently. This partitioning must be carried out with respect to each processor/resource $r \in$ RES. In particular, let $ST_r \equiv \{i : i \in \mathcal{S}, r \in R_i\}$ be the set of tasks which need resource r. Partition the task in ST_r into a sequence of smaller subsets $P_{r1} \prec P_{r2} \prec \cdots \prec P_{rm}$ such that the latest completion time of every task in subset P_{rl} is less than the earliest start time of every task in any subset P_{rk}, $k > l$.

More formally, P_{r1}, P_{r2}, ..., P_{rm} are such that: (i) $ST_r = \bigcup_{l=1}^{m} P_{rl}$, (ii) $P_{rk} \cap P_{rl} = \emptyset$ for all $k \neq l$, and (iii) $\max_{i \in P_{rk}} L_i \leq \min_{j \in P_{rl}} E_j$, for all $l > k$.

An algorithm to identify such a sequence is formally described in Figure 4. Basically, the algorithm considers the tasks in the increasing order of their earliest start times. At an intermediate step, the task under consideration (say i) is added to a subset P_{rk} if $E_i \leq \max_{\{j \in P_{rk}\}} L_j$. Otherwise, a new subset $P_{r(k+1)}$ is created and task i is added to it.

6 Resource Lower Bound Analysis

In this section, we describe the proposed method for computing a lower bound on the number of units of a processor/resource $r \in$ RES required by the application. We begin with the following definitions.

Definition 3: The overlap of a task $i \in \mathcal{S}$ in an interval $[t_1, t_2]$, $t_1 < t_2$, denoted by $\Psi(i, t_1, t_2)$, is the minimum time that must be allocated to i in $[t_1, t_2]$ to meet the constraints of the application.

Definition 4: For any real number x, let

$$\alpha(x) = \begin{cases} x & x \geq 0 \\ 0 & x \leq 0 \end{cases} \quad \text{and} \quad \mu(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0. \end{cases}$$

Theorem 3: The overlap of a *preemptive* task $i \in S$ with the interval $[t_1, t_2]$, $t_1 < t_2$, is

$$\Psi(i, t_1, t_2) = \mu(L_i - t_1) \cdot \mu(t_2 - E_i) \cdot \min \left\{ \begin{array}{l} C_i, \\ \alpha(C_i - (t_1 - E_i)), \\ \alpha(C_i - (L_i - t_2)), \\ \alpha(C_i - (L_i - t_2) - (t_1 - E_i)) \end{array} \right\} \tag{6.1}$$

Proof: To prove the theorem, we individually consider the five cases illustrated in Figure 5. In each case, we show that the right hand side (RHS) of Equation 6.1 evaluates to the minimum overlap of a preemptive task.

Case 1: $L_i \leq t_1$ or $t_2 \leq E_i$ (see Figure 5(a)).
In this case, the execution of task i cannot overlap with the interval $[t_1, t_2]$. Hence, $\Psi(i, t_1, t_2) = 0$. Note that, the RHS of Equation 6.2 also evaluates to zero because $\mu(L_i - t_1) \cdot \mu(t_2 - E_i) = 0$.

Case 2: $t_1 \leq E_i \leq L_i \leq t_2$ (see Figure 5(b)).
In this case, task i must completely execute in the interval $[t_1, t_2]$. Therefore, $\Psi(i, t_1, t_2) = C_i$. Evaluating RHS of Equation 6.2,

$$\begin{aligned} \text{RHS} &= 1 \cdot \min \{C_i, \ C_i + (E_i - t_1), \ C_i + (t_2 - L_i), \ \alpha(C_i + (t_2 - t_1) - (L_i - E_i))\} \\ &= \min \{C_i, \ C_i + (E_i - t_1), \ C_i + (t_2 - L_i), \ C_i + (t_2 - t_1) - (L_i - E_i)\} \\ &= C_i \\ &= \Psi(i, t_1, t_2). \end{aligned}$$

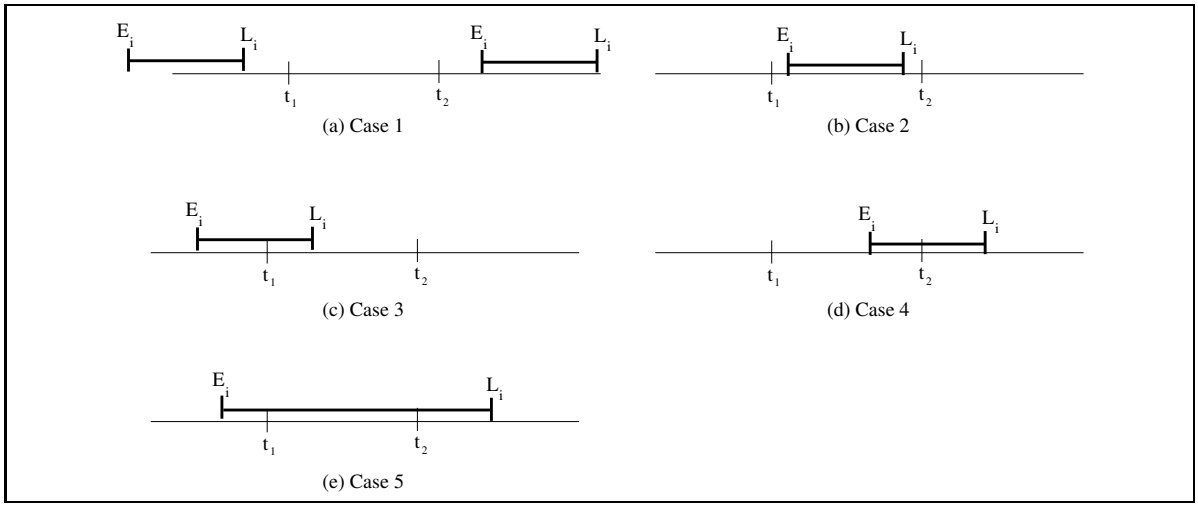

Figure 5: All possible overlap cases between $[E_i, L_i]$ and $[t_1, t_2]$.

Case 3: $E_i \leq t_1 \leq L_i \leq t_2$ (see Figure 5(c)).
To minimize the overlap with the interval $[t_1, t_2]$, task i should be executed as early as possible. Thus, the overlap of i is non-zero only if $C_i > (t_1 - E_i)$. Hence, $\Psi(i, t_1, t_2) = \alpha(C_i - (t_1 - E_i))$. Evaluating RHS of Equation 6.1,

$$\begin{aligned}
\text{RHS} &= 1 \cdot \min\{C_i,\ \alpha(C_i - (t_1 - E_i)),\ \alpha(C_i - (L_i - t_2)),\ \alpha(C_i - (L_i - t_2) - (t_1 - E_i))\} \\
&= \min\{\alpha(C_i - (t_1 - E_i)), \alpha(C_i - (t_1 - E_i) - (L_i - t_2))\} (\text{since } t_1 \geq E_i,\ t_2 \geq L_i) \\
&= \alpha(C_i - (t_1 - E_i)) (\text{since } t_2 \geq L_i) \\
&= \Psi(i, t_1, t_2).
\end{aligned}$$

Case 4: $t_1 \leq E_i \leq t_2 \leq L_i$ (see Figure 5(d)).
To minimize the overlap with the interval $[t_1, t_2]$, task i should be executed as late as possible. Thus, the overlap of i is non-zero only if $C_i > (L_i - t_2)$. Hence, $\Psi(i, t_1, t_2) = \alpha(C_i - (L_i - t_2))$. Evaluating RHS of Equation 6.1,

$$\begin{aligned}
\text{RHS} &= 1 \cdot \min\{C_i,\ \alpha(C_i - (t_1 - E_i)),\ \alpha(C_i - (L_i - t_2),\ \alpha(C_i - (L_i - t_2) - (t_1 - E_i))\} \\
&= \min\{\alpha(C_i - (L_i - t_2)), \alpha(C_i - (L_i - t_2) - (t_1 - E_i))\} (\text{since } t_1 \leq E_i,\ t_2 \leq L_i) \\
&= \alpha(C_i - (L_i - t_2)) (\text{since } t_1 \leq E_i) \\
&= \Psi(i, t_1, t_2).
\end{aligned}$$

Case 5: $E_i \leq t_1 \leq t_2 \leq L_i$ (see Figure 5(e)).
Since i is preemptive, i must (partially) execute in $[t_1, t_2]$ only if

$$C_i > (t_1 - E_i) + (L_i - t_2).$$

This is because if $C_i \leq (t_1 - E_i) + (L_i - t_2)$, then i can first execute in $[E_i, t_1]$, get preempted, and then complete its execution in $[t_2, L_i]$. Therefore,

$$\Psi(i, t_1, t_2) = \alpha(C_i - (t_1 - E_i) - (L_i - t_2)).$$

Evaluating RHS of Equation 6.1,

$$\begin{aligned}
\text{RHS} &= 1 \cdot \min\{C_i,\ \alpha(C_i - (t_1 - E_i)),\ \alpha(C_i - (L_i - t_2)),\ \alpha(C_i - (t_1 - E_i) - (L_i - t_2))\} \\
&= \min\{\ \alpha(C_i - (t_1 - E_i)),\ \alpha(C_i - (L_i - t_2)),\ \ \alpha(C_i - (t_1 - E_i) - (L_i - t_2))\} \\
&\quad (\text{since } t_1 \geq E_i,\ L_i \geq t_2) \\
&= \alpha(C_i - (t_1 - E_i) - (L_i - t_2)) \\
&= \Psi(i, t_1, t_2). \square
\end{aligned}$$

The overlap of a non-preemptive task is given by Theorem 4. The proof of the theorem is similar to that of Theorem 3 and is included in the appendix.

Theorem 4: The overlap of a *non-preemptive* task $i \in S$ in the interval $[t_1, t_2]$, $t_1 \leq t_2$, is

$$\Psi(i, t_1, t_2) = \mu(L_i - t_1) \cdot \mu(t_2 - E_i) \cdot \min \left\{ \begin{array}{l} C_i, \\ \alpha(C_i - (t_1 - E_i)), \\ \alpha(C_i - (L_i - t_2)), \\ (t_2 - t_1). \end{array} \right\} \tag{6.2}$$

Using either Theorem 3 or Theorem 4, we can determine the minimum computation time required from resource $r \in \text{RES}$ for task i in any given interval $[t_1, t_2]$. Then, the minimum computation time required from resource r for the entire application in the interval $[t_1, t_2]$ is

$$\Theta(r, t_1, t_2) = \sum_{i \in ST_r} \Psi(i, t_1, t_2).$$

Therefore, a lower bound on the number of resource units of r required for the entire application is

$$LB_r = \left\lceil \max_{\tau_s(r) \leq t_1 \leq t_2 \leq \tau_f(r)} \left\{ \frac{\Theta(r, t_1, t_2)}{t_2 - t_1} \right\} \right\rceil,\ r \in RES, \tag{6.3}$$

where $\tau_s(r) = \min_{i \in ST_r}\{E_i\}$ and $\tau_f(r) = \max_{i \in ST_r}\{L_i\}$.

The exact evaluation of Equation 6.3 is computationally infeasible because we need to consider an infinite number of intervals. To make it computationally feasible, we can find a sequence of points $a_0, a_1, \ldots, a_N$ such that $\tau_s(r) = a_0 < a_1 < a_2 \cdots < a_{N-1} < a_N = \tau_f(r)$. One can then estimate the right hand side of Equation 6.3 by evaluating only over the intervals generated by a_l, $1 \leq l \leq N$, i.e.,

$$LB'_r = \left\lceil \max_{1 \leq l < k \leq N} \left\{ \frac{\Theta(r, a_l, a_k)}{a_k - a_l} \right\} \right\rceil .$$

Note that, LB'_r is a weaker bound than LB_r because $LB_r \geq LB'_r$.

The above method is $O(N^2)$ because all possible intervals generated by the N points are considered. However, if the application is partitioned into a sequence of smaller subsets $P_{r1} \prec P_{r2} \prec \cdots \prec P_{rm}$, then it is not necessary to consider all the N^2 intervals. This reduction is characterized by the following lemma and theorem.

Lemma 1: For $1 \leq i \leq n$, $a_i \geq 0$, $b_i > 0$,

$$\frac{\sum_{i=1}^{n} a_i}{\sum_{i=1}^{n} b_i} \leq \max_{1 \leq i \leq n} \left\{ \frac{a_i}{b_i} \right\} .$$

Proof: Follows from algebraic manipulations. □

Theorem 5: Let $P_{r1} \prec P_{r2} \prec \cdots \prec P_{rm}$ be the partition of the tasks in ST_r identified by the algorithm in Figure 4. Also, let $s_k = \min\{E_i : i \in P_{rk}\}$ and $f_k = \max\{L_i : i \in P_{rk}\}$ for $1 \leq k \leq m$. Further, let

$$\Omega(r) = \{[s_k, f_k] : 1 \leq k \leq m\}.$$

Then,

$$\max_{\tau_s(r) \leq t_1 \leq t_2 \leq \tau_f(r)} \left\{ \frac{\Theta(r, t_1, t_2)}{t_2 - t_1} \right\} = \max_{\{[s_k, f_k] \in \Omega(r)\}} \left\{ \max_{\{s_k \leq t_1 \leq t_2 \leq f_k\}} \left\{ \frac{\Theta(r, t_1, t_2)}{t_2 - t_1} \right\} \right\} \qquad (6.4)$$

Proof: An interval $[t_1, t_2] \subset [\tau_s(r), \tau_f(r)]$ can be classified into one of the following three distinct types, (see Figure 6).

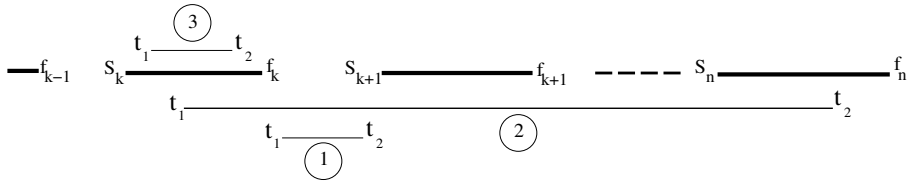

Figure 6: Interval types

- Type 1: $\exists k$, such that $f_k \leq t_1 < t_2 \leq s_{k+1}$
- Type 2: $\exists k$, such that $s_k \leq t_1 < s_{k+1} \leq t_2$, and
- Type 3: $\exists k$, such that $f_{k-1} \leq t_1 < t_2 \leq s_{k+1}$

Then,

$$\max_{\tau_s(r) \leq t_1 \leq t_2 \leq \tau_f(r)} \left\{ \frac{\Theta(r, t_1, t_2)}{t_2 - t_1} \right\} = \max \left\{ LB_r^1, LB_r^2, LB_r^3 \right\}, \tag{6.5}$$

where,

$$LB_r^j = \max_{[t_1, t_2] \text{ of Type } j} \left\{ \frac{\Theta(r, t_1, t_2)}{t_2 - t_1} \right\}, \quad 1 \leq j \leq 3.$$

Case 1: $[t_1, t_2]$ is of Type 1.
In this case, there are no tasks which can execute in $[t_1, t_2]$. Hence, $\Theta(r, t_1, t_2) = 0$ and $LB_r^1 = 0$. Since $LB_r^3 \geq 0$, it follows that $LB_r^1 \leq LB_r^3$.

Case 2: $[t_1, t_2]$ is of type 2.
Let $k = \arg\min\{f_l : f_l \geq t_1\}$ and let $n = \arg\min\{s_l : s_l \leq t_2\}$. Then,

$$\Theta(r, t_1, t_2) = \Theta(r, \max\{t_1, s_k\}, f_k) + \sum_{i=k+1}^{i=n-1} \Theta(r, s_i, f_i) + \Theta(r, s_n, \min\{t_2, f_n\}). \tag{6.6}$$

Since, $(t_2 - t_1) \geq (f_k - \max\{t_1, s_k\}) + \sum_{i=k+1}^{i=n-1} (f_i - s_i) + (\min\{t_2, f_n\} - s_n)$, it follows that,

$$\frac{\Theta(r, t_1, t_2)}{t_2 - t_1} \leq \frac{\Theta(r, \max\{t_1, s_k\}, f_k) + \sum_{i=k+1}^{i=n-1} \Theta(r, s_i, f_i) + \Theta(r, s_n, \min\{t_2, f_n\})}{(f_k - \max\{t_1, s_k\}) + \sum_{i=k+1}^{i=n-1} (f_i - s_i) + \min\{t_2, f_n\} - s_n)}$$

Thus, from Lemma 1 and further simplifications, we get

$$\begin{aligned}
\frac{\Theta(r,t_1,t_2)}{t_2-t_1} &\leq \max\left\{\frac{\Theta(r,\max\{t_1,s_k\},f_k)}{f_k-\max\{t_1,s_k\}}, \max_{k+1\leq i\leq n-1}\frac{\Theta(r,s_i,f_i)}{(f_i-s_i)}, \frac{\Theta(r,s_n,\min\{t_2,f_n\})}{\min\{t_2,f_n\}-s_n}\right\} \\
&\leq \max_{k\leq j\leq n}\left\{\max_{s_j\leq\tau_1\leq\tau_2\leq f_j}\left\{\frac{\Theta(r,\tau_1,\tau_2)}{\tau_2-\tau_1}\right\}\right\} \\
&\leq \max_{1\leq j\leq m}\left\{\max_{s_j\leq\tau_1\leq\tau_2\leq f_j}\left\{\frac{\Theta(r,\tau_1,\tau_2)}{\tau_2-\tau_1}\right\}\right\} \\
&\leq \text{RHS of Equation 6.4.}
\end{aligned}$$

Since the above result is true for each interval $[t_1,t_2]$ of type 2, we can conclude that $LB_r^2 \leq RHS of Equation\ 6.4$. The theorem follows from Cases 1 and 2.

Case 3: $[t_1,t_2]$ of Type 3

$$\Theta(r,t_1,t_2) = \Theta(r,\max\{t_1,s_k\},\min\{t_2,f_k\}).$$

From the above equation, it follows that

$$\begin{aligned}
LB_r^3 &= \max_{[t_1,t_2]\ \text{of Type 3}}\frac{\Theta(r,t_1,t_2)}{t_2-t_1} \\
&= \max_{[t_1,t_2]\ \text{of Type 3}}\frac{\Theta(r,\max\{t_1,s_k\},\min\{t_2,f_k\})}{t_2-t_1} \\
&= \text{RHS of Equation 6.4.}
\end{aligned}$$

The theorem follows from Cases 1, 2, and 3 and from Equation 6.5. □

The above theorem implies that the P_{rj}, $1 \leq j \leq m$, generated by algorithm in Figure 4 can be treated independently and the final lower bound is simply a maximum of the results obtained for each partition.

7 Lower Bound on System Cost

In this section, we present our approach for evaluating a lower bound on the the cost of the system which can meet all the constraints of the application. As stated earlier, in the shared model, the cost of the system is assumed to be sum of the costs of the processors and resources. In the dedicated model, the cost is assumed to be sum of the individual node costs.

In the shared model, a lower bound on the cost of the system is directly obtained using the lower bound on the number of units of each resource $r \in \text{RES}$. That is, system cost

$$\text{Shared System Cost} \geq \sum_{r \in \text{RES}} \text{CostR}(r) \cdot LB_r. \tag{7.1}$$

In the dedicated model, the lower bound analysis is more involved because tasks can only be assigned to nodes which have all the resources they need. Furthermore, the analysis in Section 6 does not provide a lower bound on the number of units of each node type required by the application. Therefore, we need to estimate the required number of units of each node type using the lower bounds for the processors and the resources required by the application.

Let x_n denote the number of units of a node type n required to meet the constraints of the application. Also, let γ_{nr} be the number of units of $r \in \text{RES}$ in a node of type n. The cost of the system is then given by

$$\text{Dedicated System Cost} \geq \sum_{n \in \Lambda} \text{CostN}(n) \cdot x_n. \tag{7.2}$$

From the lower bound analysis, we know that at least LB_r units of each resource must be present in the system. Therefore, the values of x_n must satisfy the following constraint.

$$\sum_{n \in \Lambda} x_n \cdot \gamma_{nr} \geq LB_r, \quad \forall r \in \text{RES}.$$

In addition, for each task, the system must have at least one node on which the task can execute. Hence, if η_i is the set of node types on which task $i \in \mathcal{S}$ can execute, then

$$\sum_{n \in \eta_i} x_n \geq 1, \quad \forall i \in \mathcal{S}.$$

A lower bound on the cost of a dedicated system can thus be determined by solving the following optimization problem.

$$\textbf{Minimize} \sum_{n \in \Lambda} x_n \cdot \text{CostN}(n)$$

Subject to:

$$\sum_{n \in \Lambda} x_n \cdot \gamma_{nr} \geq LB_r, \quad \forall r \in \text{RES}$$

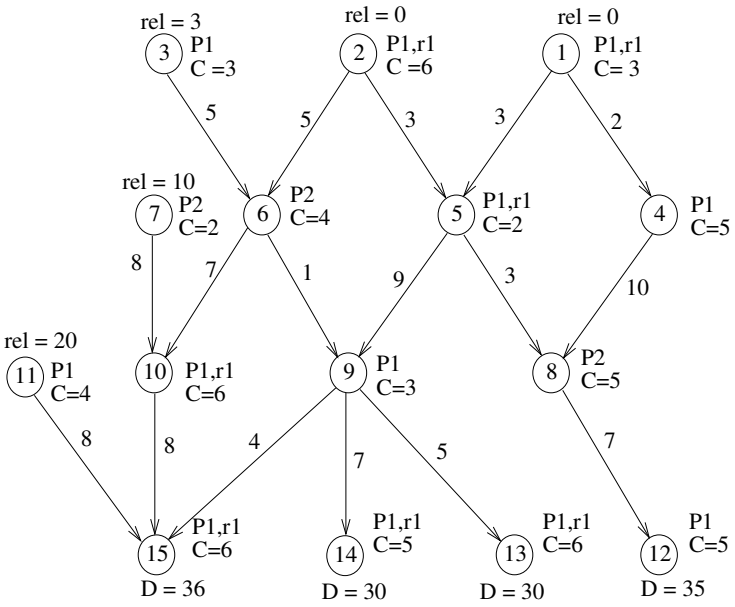

Figure 7: Example of a real-time application.

$$\sum_{n \in \eta_i} x_n \geq 1, \quad \forall i \in \mathcal{S}.$$

In the above problem, x_n is an integer for each $n \in \Lambda$. If we relax this requirement, the resulting cost will also be a lower bound. However, the bound will be weaker than the one obtained if we solve it as an integer programming problem.

8 Illustrative Example

In this section, we present a simple example to illustrate the steps involved in the evaluation of the lower bounds. Consider the example in Figure 7. The figure shows 15 tasks numbered 1–15. Each task is annotated with its computation time and its resource requirement. Tasks 1, 2, 3, 7, 11 are also annotated with their release times. The release time of the other tasks is assumed to be 0. Tasks 12–15 are annotated with their deadlines. The deadline of other tasks is assumed to be 36. All tasks are non-preemptive. There are two types of processors, P_1 and P_2, and one type of resource, r_1, required for the application, i.e., RES = $\{P_1, P_2, r_1\}$. In addition, we are also given the following node types for the dedicated model, $\Lambda = \{\{P_1, r_1\}, \{P_1\}, \{P_2\}\}$. In this example, a set of tasks which are mergeable in the *shared model* are also mergeable in the *dedicated* model. In the following we consider the four steps of the algorithm.

Table 1: EST and LST

Task i	E_i	M_i	L_i	G_i
1	0	–	3	{4}
2	0	–	6	–
3	3	–	6	–
4	3	{1}	8	–
5	6	{2}	15	{9}
6	11	–	15	–
7	10	–	16	–
8	18	–	23	–
9	16	{5}	19	{14,13}
10	22	–	30	{15}
11	20	–	35	{15}
12	30	–	30	–
13	19	{9}	30	–
14	19	{9}	30	–
15	30	{10,11}	36	–

Step 1: Evaluation of LCT and EST (refer to Figures 2 and 3)
The values of E_i and L_i are shown in Table 1. The table also shows M_i, the set of predecessors merged with task i in evaluating E_i. Similarly, the table also shows G_i, the set of successors merged with i in evaluating L_i. A "–" is used to indicate that no tasks are merged with i.

Consider the evaluation of L_9. The values of $lms_j, j \in Succ_9$ are $\text{lms}_{15} = L_{15} - C_{15} - m_{9,15} = 36 - 6 - 4 = 26$, $\text{lms}_{14} = 30 - 5 - 7 = 18$, and $\text{lms}_{13} = 30 - 6 - 5 = 19$. If no tasks are merged with task 9, then, its LCT will be 18. To increase its LCT, the algorithm first evaluates the effect of merging tasks 14 and 9; task 14 is chosen because it has the least latest message send time. If these two tasks are merged, the LCT of task 9 will become $L_9^1 = \min\{\text{lst}(\{14\}), \text{lms}_{13}, \text{lms}_{15}\} = \min\{25, 19, 26\} = 19$. Since this is greater than 18, tasks 14 and 9 will be merged together. Next, task 13 is considered for merging. Merging task 13 with $\{9, 14\}$ will result in $L_9^2 = \min\{\text{lst}(\{14, 13\}), \text{lms}_{15}\} = \min\{19, 26\}$. The algorithm stops here because $L_9^2 = L_9^1$, and thus merging more tasks will only decrease the LCT of task 9. Thus, $L_9 = 19$.

As another example, consider the evaluation of L_5. Note that, at this time, L_9 and L_8 have already been evaluated. In this case, $lms_9 = 19 - 3 - 9 = 7$ and $lms_8 = 23 - 5 - 3 = 15$. Therefore, task 9 is first considered for merging. This results in $L_5^1 = \min\{\text{lst}(\{9\}, \text{lms}_8) = 15$. Next, task 8 is considered for merging. However, task 8 cannot be merged with task 5

because it requires a different type of processor to execute. The algorithm, therefore, stops here with $L_5 = 15$.

Step 2: Partitioning (refer to Figure 4)
With respect to P_1, P_2, and r_1, the partitions generated by the algorithm in Figure 4 are $ST_{P_1} = \{1,2,3,4,5\} \prec \{9\} \prec \{10,11,13,14\}, \prec \{12,15\}$, $ST_{P_2} = \{6,7\} \prec \{8\}$, and $ST_{r_1} = \{1,2\} \prec \{5\} \prec \{10,13,14\} \prec \{15\}$.

Step 3: Lower bound evaluation (refer to Theorem 5)
As a result of the partitioning in Step 2, LB_{P_1} is evaluated over the intervals [0, 15], [16, 19], [19, 30], and [30, 36]. Likewise, LB_{P_2} is evaluated over [10, 16] and [18, 23] and LB_{r_1} is evaluated over [0, 6], [6, 15], [19, 30], and [30, 36].

In each interval, we must select few points and evaluate the lower bound over the intervals generated by these points. Suppose the selected points are those generated by ESTs and the LCTs which lie in the interval. For example, consider the intervals generated by {0, 3, 6, 8, 16} in the interval [0, 15] for evaluation LB_{P_1}. Here, $\lceil \frac{\Theta(P_1,0,3)}{3} \rceil = \lceil \frac{6}{3} \rceil = 2$, $\lceil \frac{\Theta(P_1,3,6)}{3} \rceil = \lceil \frac{9}{3} \rceil = 3$, $\lceil \frac{\Theta(P_1,3,8)}{5} \rceil = \rceil \frac{11}{5} = 3$. and so on. By taking the maximum over all such intervals we get, $LB_{P_1} = 3$, $LB_{P_2} = 2$, and $LB_{r_1} = 2$.

Step 4: Cost evaluation (refer to Section 7)
For the shared model, a lower bound on the cost of the system is

$$\text{Shared System Cost} \;=\; 3 \cdot \text{CostR}(P_1) + 2 \cdot \text{CostR}(P_2) + +2 \cdot \text{CostR}(r_1)$$

For the dedicated model, let x_1 be the units of node of type $\{P_1, r_1\}$, x_2 be the units of node of type $\{P_1\}$, and x_3 be the units of node of type $\{P_2\}$. We can obtain a lower bound on the cost of the dedicated model by solving the following optimization problem.

$$\textbf{Minimize} \quad x_1 \cdot \text{CostN}(1) + x_2 \cdot \text{CostN}(2) + x_3 \cdot \text{CostN}(3)$$

subject to

$$\begin{aligned} x_1 + x_2 &\geq 3, \\ x_1 &\geq 2, \text{ and} \\ x_3 &\geq 2. \end{aligned}$$

The solution to this problem is, $x_1 = 2, x_2 = 1, x_3 = 2$. Hence,

$$\text{Dedicated System Cost} \;=\; 2 \cdot \text{CostN}(1) + \text{CostN}(2) + 2 \cdot \text{CostN}(3)$$

9 Conclusions

Given a real-time application and a model for the distributed computing system, we proposed a technique to estimate the number of processors and resources of each type required to meet the constraints of the application. We proved that this estimate is a lower bound on the number of processors and resources necessary to meet the application constraints. Using these bounds, we also proposed a scheme to determine a lower bound on the cost of a distributed system which meets all the application constraints.

The key feature of the proposed analysis technique is that it can handle several different types of constraints commonly found in real-time applications. In particular, it can deal with real-time constraints such as deadlines and release times and non-real-time constraints like precedence relationships between tasks, communication needs, and resource requirements. In addition, the application tasks can be heterogeneous in the sense that they may differ on the type of processor which can execute them. Furthermore, each task can be either preemptive or non-preemptive. To the best of our knowledge, none of the existing techniques can deal with applications which have all these constraints.

The paper also considers two possible architectures of distributed computing systems. In one architecture, the resources in the system are shared by all processors, whereas in the other, each processor has its own set of dedicated resources. A designer can apply the proposed technique to these architectures and choose the one that is best suited for the application. For instance, a designer can modify the set of resources dedicated to a processor and quickly estimate its effect on the overall system cost. As a result, designers can explore more alternatives and obtain a better system for their application.

References

[1] M. A. Al-Mohummed, "Lower bound on the number of processors and time for scheduling precedence graphs with communication costs," *IEEE Transactions on Software Engineering*, vol. 16, no. 12, pp. 1390–1401, December 1990.

[2] R. Alqadi and P. Ramanathan, "Architectural synthesis of mission-critical computing systems," in *Proceedings Complex Systems Engineering Synthesis and Assessment Technology Workshop*, pp. 185–192. Naval Surface Warfare Center, Silver Spring, Maryland, July 1993.

[3] E. B. Fernandez and B. Bussell, "Bounds on the number of processors and time for multiprocessor optimal schedules," *IEEE Transactions on Computers*, vol. C-22, no. 8, pp. 745–751, August 1973.

[4] S. Howell, C. M. Nguyen, and P. Q. Hwang, "System design structuring and allocation optimization," in *Proceedings of the 1991 Systems Design Synthesis Technology Workshop*, pp. 117–128, September 1991.

[5] K. K. Jain and V. Rajaraman, "Lower and upper bounds on time for multiprocessor optimal schedules," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 8, pp. 879–886, August 1994.

[6] B. P. Lester, "A system for computing the speedup of parallel programs," in *Proceedings of International Conference on Parallel Processing*, pp. 145–152, August 1986.

[7] J. W. S. Liu, J.-R. Redondo, Z. Deng, T.-S. Tia, R. Bettati, A. Silberman, M. S. an R. Ha, and W.-K. Shih, "PERTS: A prototyping environment for real-time systems," Technical report UIUCDCS-R-93-1802, University of Illinois, Urbana, Illinois, May 1993.

[8] J. J. Molini, S. K. Maimon, and P. H. Watson, "Real-time system scenarios," in *Proceedings of Real-Time Systems Symposium*, pp. 214–225, December 1990.

[9] C. Polychronopoulos and U. Banerjee, "Speedup bounds and processors allocation for parallel programs on multiprocessors," in *Proceedings of International Conference on Parallel Processing*, pp. 961–968, August 1986.

[10] K. So et al., "A speedup analyzer for parallel programs," in *Proceedings of International Conference on Parallel Processing*, pp. 653–661, August 1987.

[11] X.-H. Sun and L. M. Ni, "Another view on parallel speedup," in *Proceedings of the 3rd Supercomputing Conference*, pp. 324–333, 1990.

APPENDIX

A. Proof of Theorem 2

Proof: Consider a task i. Let E be earliest start time of i returned by the algorithm. Also, let M be the corresponding set of predecessors of i which are merged with i in the algorithm. Let $\bar{M} = \text{Pred}_i - M$. We now show that if task i cannot start earlier than E.

From the algorithm, we know that

$$E = \max\{\text{rel}_i, \ \text{ect}(M), \ \max_{j \in \bar{M}} \text{emr}_j\} \ \text{and} \tag{.1}$$

$$\text{emr}_j \geq E \quad \forall j \in M \tag{.2}$$

Consider a $M' \neq M$. Let $\bar{M}' = \text{Pred}_i - M'$. Let E' be the earliest start time of i if only the tasks in M' are merged with i. Then,

$$E' = \max\{\text{rel}_i, \ \text{ect}(M'), \ \max_{j \in \bar{M}'} \text{emr}_j\}. \tag{.3}$$

We must prove that $E' \geq E$.

Case 1: $M \not\subset M'$
Consider a task $T \in \bar{M'} \cap M$. Hence, from Equations .3 and .2, $E' \geq \text{emr}_T \geq E$.

Case 2: $M \subseteq M'$
Let $T = \arg\max_{j \in \bar{M}} \text{emr}_j$, i.e., T is the value of j which corresponds to the maximum emr_j. Now consider two cases.

Case 2a: $T \in M'$.
Since $T \in M'$, T is mergeable with M. Therefore, it follows from the algorithm that $\text{ect}(M \cup \{T\}) \geq E$. The result then follows because

$$\begin{aligned} E' &\geq \text{ect}(M') \quad \text{(from Equation .3)} \\ &\geq \text{ect}(M \cup \{T\}) \quad \text{(since } M \cup \{T\} \subseteq M') \\ &\geq E. \end{aligned}$$

Case 2b: $T \notin M'$.
From Equation .1 and the definition of T,

$$E = \max\{\text{rel}_i, \text{ect}(M), \text{emr}_T\}. \tag{.4}$$

Similarly, from Equation .3 and the definition of T,

$$E' = \max\{\text{rel}_i, \text{ect}(M'), \text{emr}_T\}. \tag{.5}$$

Since $M \subseteq M'$, $\text{ect}(M') \geq \text{ect}(M)$. Therefore, from Equations .4 and .5, we can conclude that $E' \geq E$.

From Cases 1, 2a, 2b, we can conclude that the subset M identified by the algorithm results in the smallest earliest start time for i. Hence, the algorithm evaluates a lower bound on the earliest start time of each task in the application. □

B. Proof of Theorem 4

Proof: To prove the theorem, we individually consider the five cases illustrated in Figure 5. In each case, we show that the right hand side (RHS) of Equation 6.2 evaluates to the minimum overlap.

Case 1: $L_i \leq t_1$ or $t_2 \leq E_i$ (see Figure 5(a)).
In this case, the execution of task i cannot overlap with the interval $[t_1, t_2]$. Hence,

$\Psi(i, t_1, t_2) = 0$. Note that, the RHS of Equation 6.2 also evaluates to zero because $\mu(L_i - t_1) \cdot \mu(t_2 - E_i) = 0$.

Case 2: $t_1 \leq E_i \leq L_i \leq t_2$ (see Figure 5(b)).
In this case, task i must completely execute in the interval $[t_1, t_2]$. Therefore, $\Psi(i, t_1, t_2) = C_i$. Evaluating RHS of Equation 6.2,

$$\begin{aligned} \text{RHS} &= 1 \cdot \min\{\{C_i, \ \ C_i + (E_i - t_1), \ \ C_i + (t_2 - L_i), \ \ (t_2 - t_1)\} \\ &= C_i \\ &= \Psi(i, t_1, t_2). \end{aligned}$$

Case 3: $E_i \leq t_1 \leq L_i \leq t_2$ (see Figure 5(c)).
To minimize the overlap with the interval $[t_1, t_2]$, task i should be executed as early as possible. Thus, the overlap of i is non-zero only if $C_i > (t_1 - E_i)$. Hence, $\Psi(i, t_1, t_2) = \alpha(C_i - (t_1 - E_i))$. Evaluating RHS of Equation 6.2,

$$\begin{aligned} \text{RHS} &= 1 \cdot \min\{C_i, \ \ \alpha(C_i - (t_1 - E_i)), \alpha(C_i - (L_i - t_2)), \ \ (t_2 - t_1)\} \\ &= \min\{C_i, \ \ \alpha(C_i - (t_1 - E_i)), \ \ (t_2 - t_1)\} (\text{since } t_2 \geq L_i) \\ &= \min\{\alpha(C_i - (t_1 - E_i)), \ \ (t_2 - t_1)\} (\text{since } t_1 \geq E_i) \\ &= \alpha(C_i - (t_1 - E_i)) (\text{since } \alpha(C_i - (t_1 - E_i)) \leq (L_i - t_1) \leq (t_2 - t_1)) \\ &= \Psi(i, t_1, t_2). \end{aligned}$$

Case 4: $t_1 \leq E_i \leq t_2 \leq L_i$ (see Figure 5(d)).
To minimize the overlap with the interval $[t_1, t_2]$, task i should be executed as late as possible. Thus, the overlap of i is non-zero only if $C_i > (L_i - t_2)$. Hence, $\Psi(i, t_1, t_2) = \alpha(C_i - (L_i - t_2))$. Evaluating RHS of Equation 6.2,

$$\begin{aligned} \text{RHS} &= 1 \cdot \min\{C_i, \ \ \alpha(C_i - (t_1 - E_i)), \alpha(C_i - (L_i - t_2)), \ \ (t_2 - t_1)\} \\ &= \min\{C_i, \ \ \alpha(C_i - (L_i - t_2)), \ \ (t_2 - t_1)\} (\text{since } t_1 \leq E_i) \\ &= \min\{\alpha(C_i - (L_i - t_2)), \ \ (t_2 - t_1)\} (\text{since } L_i \geq t_2) \\ &= \alpha(C_i - (L_i - t_2)) (\text{since } \alpha(C_i - (L - i - t_2)) \leq (t_2 - E_i) \leq (t_2 - t_1)) \\ &= \Psi(i, t_1, t_2). \end{aligned}$$

Case 5: $E_i \leq t_1 \leq t_2 \leq L_i$ (see Figure 5(e)).
Minimum overlap occurs in one of the following two cases; i is executed as early as possible

or i is executed as late as possible. That is, $\Psi(i, t_1, t_2) = \min\{\alpha(C_i - (t_1 - E_i)), \alpha(C_i - (L_i - t_2)), (t_2 - t_1)\}$.

Evaluating RHS of Equation 6.2,

$$\begin{aligned} \text{RHS} &= 1 \cdot \min\{\alpha(C_i - (t_1 - E_i)), \alpha(C_i - (L_i - t_2)), \ (t_2 - t_1)\} \quad (\text{since } t_1 \geq E_i \text{ and } L_i \geq t_2) \\ &= \Psi(i, t_1, t_2). \end{aligned}$$

□