Proceedings, IEEE International Conference on Engineering of Complex Computer Systems, Oct. 1996

Resource Conscious Design of Distributed Real-Time Systems An End-to-End Approach*

Manas Saksena
Concordia University

Seongsoo Hong
Seoul National University

Abstract

In this paper, we present a resource conscious approach to designing distributed real-time systems. This work extends our original solution [6], which was limited to single processor systems. Starting from a given task graph, and a set of end-to-end constraints, we systematically generate task attributes (e.g., periods and deadlines) such that (i) the task set is schedulable, and (ii) the end-to-end constraints are satisfied. The methodology presented in this paper can be mostly automated, and provides useful feedback to a designer when it fails to find a solution. We expect that the techniques presented in this paper will help reduce the laborious process of designing a real-time system, by bringing resource contention and schedulability aspects early into the design process.

1 Introduction

Recent developments have resulted in the maturity of real-time scheduling theory techniques, and shown the viability of these techniques for industrial real-time systems especially in the domain of distributed control systems [3, 9]. However, while this progress is admirable, most real-time scheduling techniques handle timing constraints (i.e., task periods, deadlines, etc.) that are artifacts of system design, and for a given system may be chosen in many different ways to meet the end user requirements. Consider, for example, a robotic control system [15]. The control laws are generally implemented as multi-task programs, and may be allocated to multiple processors on a distributed system. The timing behavior of such a system must ensure that end user requirements such as maximum value of tracking errors are satisfied. Such properties intimately depend on factors such as sampling rates for inputs, update rates for outputs, and end-to-end latencies from sensor to actuators [8]. Through simulation or analysis, a control engineer can specify requirements such as maximum acceptable sampling rates for a given control loop, or maximum latency from a sensor input to an actuator output, and these requirements constitute what we call high-level end-to-end timing constraints on system inputs and outputs.

The process of deriving task attributes from such high level timing requirements is perhaps the most ad-hoc of all steps in the development process of a real-time system. In current engineering practice, task attributes are often mandated by control engineers and rarely take into account the resource constraints. As systems become more complex, such ad-hoc methods to derive feasible task parameters do not scale well due to to the manual and labor-extensive process of trial and error based on engineering intuition. In addition, many of the synchronization requirements get tightly coupled with the derived timing constraints. This loss in traceability of requirements for a system under development may result in significant redesigns when timing constraints are changed.

In this paper we address the problem of transforming a high-level real-time system design into a set of schedulable periodic tasks. The current work improves and extends our original solution [6] for a single processor system to a distributed system environment. We model the problem as a constraint solving problem, in which the original end-to-end timing constraints are expressed as a set of constraints on task attributes. The constraints are then solved to derive a set of schedulable task parameters. Figure 1 shows an overview of our methodology. As shown in the figure, our objective is to generate an implementable set of tasks from a task graph design of the system. This involves (1) deriving task periods, deadlines, phases, etc., and (2) synthesizing code for inter-task communication. In this paper, we focus our attention on the constraint derivation and constraint solving aspects of our design methodology.

The main contribution of our work is the develop-

*The work reported in this paper was supported in part by NSERC Operating Grant OGP0170345, and by Engineering Research Center for Advanced Control and Instrumentation (ERC-ACI) under Grant 95-26.

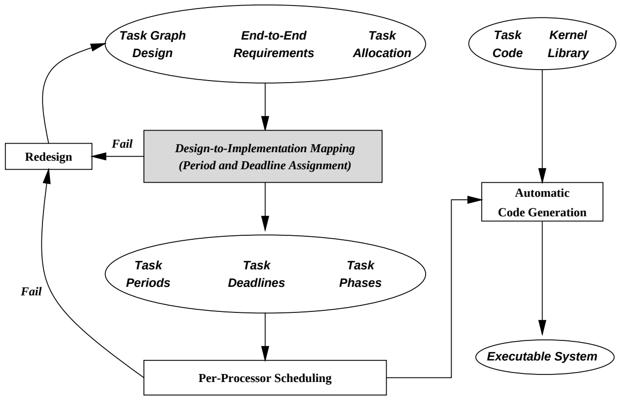

Figure 1. End-to-End Design Methodology

ment of a systematic methodology to transform a high-level design into a schedulable system. The methodology presented in this paper provides substantial benefits: (1) It provides designers with a rapid prototyping tool which helps them build a running prototype quickly, and to locate and isolate schedulability and performance bottlenecks, and (2) It helps the designers fix and optimize a faulty design for both correctness and performance. This is possible not only because the system traceability is maintained throughout the approach, but also because the constraint solver itself generates various performance metrics.

1.1 Related Work

Real-time system design and scheduling have been fertile areas of research in the last decade. We refer the readers to [7] for an overview of design methods and [3, 12] for an overview of real-time scheduling. There has been relatively less effort in the integration of design and scheduling, and specifically the derivation of task periods and deadlines from end-to-end constraints. In [1] and [14] similar problems are addressed, but the focus was more on schedulability analysis, and less on the derivation of task parameters. There have also been some studies on the decomposition of end-to-end deadlines into local task deadlines [16, 2, 5, 11]. Recently, [13] reports a study on deriving task periods based on control performance metrics.

2 Problem Formulation

In this section, we first present the system and network model. Then, we formulate a system design model in terms of a task graph and end-to-end timing constraints. We also present the implementation model consisting of a set of periodic tasks and task specific attributes. Finally, we summarize our problem and give a solution overview.

2.1 System and Network Model

We consider a distributed system of processing hosts, sensors, and actuators connected together by a real-time communication network such as Fieldbus. The sensors and actuators are either autonomous devices directly connected to the communications network, or may be attached to a host via the host's local bus. Each processing host is a single CPU system and has a suitable real-time operating system which can be used to implement periodic real-time tasks and perform schedulability analysis on it. Likewise, we assume that the communication network is capable of guaranteeing bounded message transfers for periodic messages.

2.2 Design Model

We use a simple producer-consumer model, similar to models proposed by other researchers (e.g., [15]) to represent a real-time system. The model incorporates many essential features such as task sharing, simple synchronization, software reusability, network transparent communication, etc. In this model, a real-time system is composed of a set of communicating tasks. A task executes by reading data from all its input ports, operating on the data, and finally writing data to its output ports. Ports provide a network transparent message passing abstraction, and have a single writer restriction. The writes to a port are always asynchronous and non-blocking. On the other hand, the reads are synchronous, that is a process reading from a port waits for data to be written.

Sensors and Actuators form the external interface of the system with the environment. We assume that each sensor is read by a special task, which reads the data from the sensor and writes it to a port to be read by computation tasks. Likewise, there is an actuator task for each actuator which reads data from an input port and sends commands to the actuator. When a sensor or an actuator is an autonomous device, it is treated as a special processor with a single sensor or actuator task.

A real-time system designed as above may be represented as a finite, directed, acyclic graph where the tasks and the ports form the nodes in the graph, and the edges correspond to reads and writes to ports. In such a design the path from a sensor to an actuator forms a chain of producer-consumer pairs forming an end-to-end computation. Timing constraints are often defined on such end-to-end computations. We use the term *transaction* for end-to-end computations, which is defined as a relation between sensors and actuators. Let $\Gamma(\mathcal{S}\|\mathcal{A})$ represent a transaction that takes inputs from sensors in set $\mathcal{S}$, and produces output for actuators in set $\mathcal{A}$. The transaction then consists of all the tasks that fall on the path from any sensor in $\mathcal{S}$ to any actuator in $\mathcal{A}$.

End-to-End Timing Constraints. The following types of timing constraints are allowed to be established on the transactions.

(1) *Maximum Allowable Validity Time:* This constraint ensures that a data sample is used before it loses its freshness, and thus, it bounds the maximum end-to-end delay permissible between the reading of a sensor and the delivery of the output command to an actuator based on that reading. We use the notation $M(S\|A)$ to represent this maximum delay from sensor S to actuator A.

(2) *Input Data Synchronization:* This constraint ensures that when multiple sensors collaborate in driving an actuator, then the maximum time-skew between the sensor readings is bounded. We use the notation $Sync(S_1, S_2\|A)$ to denote the maximum time-skew between two sensors S_1 and S_2. Let t_1 and t_2 be the time points when two sensors S_1 and S_2 are read to drive A, respectively. Then we have

$$|t_1 - t_2| \leq Sync(S_1, S_2\|A).$$

(3) *Maximum Transaction Period:* This constraint bounds the maximum activation period for an end-to-end computation. We use the notation $MaxP(S\|A)$ to denote the maximum activation period.

2.3 Implementation Model

We assume that the real-time system is implemented as a set of periodic tasks. A periodic task τ_i is represented by a 5-tuple $\langle e_i, T_i, d_i, \phi_i, \mathcal{P}_i\rangle$, where e_i represents the task's execution time, T_i its activation period, d_i its deadline relative to the start of period, ϕ_i its initial phase (denoting the initial activation time of the periodic task) and $\mathcal{P}_i$ the processor to which it is allocated.

Data transfer over the communication network generates a set of periodic message streams. We label the message streams as m_1, m_2, m_3, etc. Each message stream m_i is treated just like a task, with the notations T_i^m, e_i^m, d_i^m, and ϕ_i^m denoting the period, the maximum message size, the relative deadline, and the initial phasing respectively. In our analysis, the network is treated as a processor, and the periodic message streams are considered as tasks. Therefore, in the remainder of this paper, we do not make any distinction between computation tasks and message streams, and whenever we refer to computation tasks, it also applies to message streams.

2.4 A Walk-Through Example

Figure 2 shows the task graph for a simple real-time system along with the allocation of tasks to hosts. The system consists of two transactions, one driving actuator A_1 and the other driving actuator A_2. There are two sensors in the system S_1 and S_2 which are read by sensor tasks τ_1 and τ_2. Likewise, the actuators are given commands by actuator tasks τ_7 and τ_8. There are four control tasks allocated on two processors. This example, which we use throughout the paper, helps us illustrate the key aspects of our methodology. The end-to-end constraints postulated on the system are given in Table 1. Task execution delays are also specified in the table; we assume that the sensor and actuator tasks take negligible execution delay.

$\mathbf{\Gamma(S_1, S_2\|A_1)}$	$\mathbf{\Gamma(S_2\|A_2)}$
$Sync(S_1, S_2\|A_1) = 1ms$	
$M(S_1, S_2\|A_1) = 40ms$	$M(S_2\|A_2) = 60ms$
$MaxP(S_1, S_2\|A_1) = 20ms$	$MaxP(S_2\|A_2) = 50ms$

$e_1 = 0ms$,	$e_2 = 0ms$,	$e_3 = 7ms$,	$e_4 = 8ms$,
$e_5 = 9ms$,	$e_6 = 15ms$,	$e_7 = 0ms$,	$e_8 = 0ms$

Table 1. Timing constraints for example.

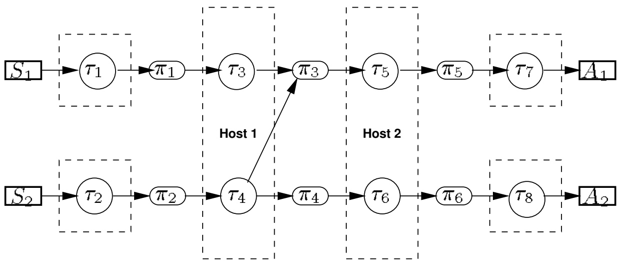

Figure 2. Task Graph Design of a Real-Time System

2.5 Problem Description

Our objective is to derive implementation parameters (i.e., the activation period, deadline, and phase) for each task and message stream from a given task graph design of a system, and the end-to-end timing requirements specified for it. We model the problem as a constraint solving problem in which the end-to-end requirements are first expressed as a set of constraints on task parameters. The intermediate constraints are then solved to derive the task parameters. The derivation of intermediate constraints preserves timing correctness, i.e., if the final task set (which is a solution of the constraints) is schedulable, then the original end-to-end requirements will be satisfied. We do not specifically address the schedulability analysis of the final task set, instead we incorporate notions of schedulability into our constraint solving process, so that the solution generated is not trivially unschedulable.

Unfortunately, solving the system of constraints is not an easy proposition due to several factors. First, in a distributed system there may be many tasks, and that will induce many variables. Second, the intermediate constraints are not always linear, as will become clear in Section 3. Third, incorporating schedulability into the constraint solving process adds significant non-linearity to the problem. To tackle this complexity, we decompose the constraint solving problem into a sequence of sub-problems. The motivation behind this approach is that each problem is more tractable, and therefore more amenable to targeted heuristics and providing useful feedback on failure. However, the success of the overall approach depends critically on whether the problem can be decomposed into well-defined sub-problems. For our case, we decompose the problem into two sub-problems *Period Assignment*, and *Phase and Deadline Assignment*, which are outlined below.

(1) *Period Assignment.* The first sub-problem that we solve is that of assigning activation periods to tasks. Since no other task parameters have been determined yet, we use overall utilization on a processor as a crude schedulability measure, which only depends on periods and execution times.

(2) *Phase and Deadline Assignment.* Once the periods are determined, we proceed to determine the phase and deadline of tasks. The main problem here is to determine a set of individual task deadlines, such that we can find some way to schedule each task within its deadlines. The phases are determined last, but we do not use any schedulability measure with them – their main purpose is to maintain synchronization and input correlation, and so any solution consistent with the constraints is acceptable.

The order of solving the two sub-problems is critically important, as it is very hard to obtain useful schedulability criteria the other way round. We defer more precise specifications of the sub-problems until later after we have shown how the entire constraint problem is setup.

3 Intermediate Constraint Derivation

The first step in our methodology is to transform the end-to-end constraints and synchronization requirements into a set of intermediate constraints on task attributes. This is a three-step process which is illustrated in the following subsections.

3.1 Synchronization and Harmonicity

The producer/consumer model, which forms the basic communication semantics in our transaction model, inherently incurs blocking synchronization for the consumer. In the periodic task model, this induces coupling between task periods which may results in unnecessarily high rates of execution for some tasks. Consider, for example, a producer task τ_p writing to a port π, which is read synchronously by two consumer tasks τ_{c_1} and τ_{c_2}. Due to blocking semantics, one would normally require equal rates of execution for consumer and producer, i.e., $T_{c_1} = T_{c_2} = T_p$. Suppose that τ_{c_1} and τ_{c_2} have maximum activation period requirements of 100ms and 350ms respectively. Due to synchronization, all tasks would need to execute at a period of 100ms, resulting in an unnecessarily high rate for τ_{c_2}, which results in wasted CPU capacity, and may make the system unschedulable. A better solution is to set $T_p = T_{c_1} = 100$, and $T_{c_2} = 300$. With this solution, we have less waste of processor capacity while preserving clean semantics for synchronization, i.e., *Task* τ_{c_2} *synchronizes with every third execution of* τ_p.

The above strategy works only when the consumer's period is an integral multiple of the producer's period. We refer this relation as *harmonic*, and this imposes a *harmonicity* constraint between any producer/consumer pair. This is represented as $T_c|T_p$, signifying that T_c is exactly divisible by T_p. Given a task graph for a system, the harmonicity constraints can be represented in a harmonicity graph, in which the task periods form the nodes, and the edges represent the harmonicity constraints. That is, an edge $T_i \longrightarrow T_j$ represents the constraint $T_j|T_i$.

3.2 Precedence and End-to-End Delay

The producer/consumer synchronization imposes a precedence requirement between producer task execution and consumer task execution. We achieve this precedence using task phasings. This avoids run-time synchronization overheads, and facilitates schedulability analysis.

For a given producer/consumer pair (τ_p, τ_c), with $T_c = k.T_p$, we require that the i^{th} iteration of task τ_c reads data produced by $k * i^{th}$ iteration of task τ_p (assuming the iteration numbers start from 0). The phase difference between the corresponding iterations is then $\phi_c - \phi_p$, the initial phase difference. Thus, the precedence is guaranteed if $\phi_c \geq \phi_p + d_p$.

Figure 3 illustrates such synchronous data communication when $T_c = 2T_p$. Two successive data transfers from the producer τ_p to the consumer τ_c using port π are shown on the timeline. We are interested in finding the maximum delay from the time τ_p reads data from its input ports, to the time τ_c writes data to its output ports. In the worst case, τ_p will read its input data just as it is invoked, while τ_c will write to the output port at its deadline. Therefore, the worst case delay becomes $\phi_c - \phi_p + d_c$. By extending the same logic to a chain of producer-consumer tasks, we can derive the end-to-end delay from a sensor task τ_s to an actuator task τ_a as $\phi_a - \phi_s + d_a$. Thus, if $M(S\|A)$ is the corresponding freshness requirement, then the following intermediate constraint must be satisfied: $\phi_a - \phi_s + d_a \leq M(S\|A)$.

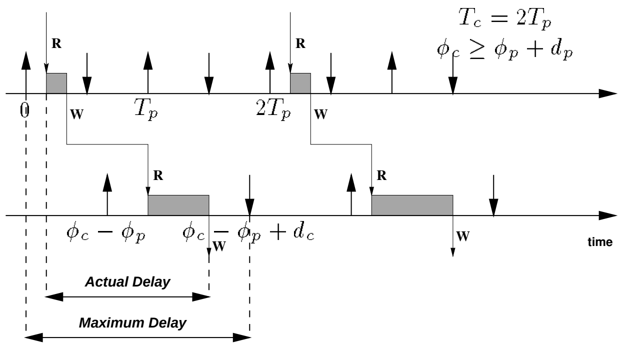

Figure 3. Delay and Precedence in Communication

3.3 Phasing and Input Synchronization

Consider an input synchronization requirement $Sync(S_1, S_2\|A)$. Let τ_1 and τ_2 denote the sensor tasks reading from sensors S_1 and S_2 respectively. Then the worst skew occurs when one task reads the sensor at its activation time, and the other at its deadline. This implies that the synchronization requirement is satisfied if: $\max(\phi_2 + d_2 - \phi_1, \phi_1 + d_1 - \phi_2) \leq Synch(S_1, S_2\|A)$. Due to harmonicity, this phase difference will be maintained for every pair of input that must be correlated even when the actual periods are different. We reduce this skew by setting $\phi_2 = \phi_1$, and thereby simplifying the constraint to: $\max(d_1, d_2) \leq Sync(S_1, S_2\|A)$. This easily generalizes to the case when we may have to synchronize 3 (or more) inputs.

3.4 Constraints for Walk-Through Example

The above procedure results in a large number of variables and constraints even for a simple system as our example. Fortunately, many of these constraints may be replaced by equalities, thereby reducing the number of free variables. One simplification occurs when a task produces data for a single consumer. In this case the producer's period can be set equal to the consumer. Another simplification occurs when a task has a single input port and, therefore, must wait for only one producer. We can then set the phase of the task to coincide with the deadline of the producer. The following table shows the derived constraints for our walk-through example as a result of applying these rules.

Constraints on Periods
$T_5 \| T_4,\ T_6 \| T_4\ T_5 \in [9,20],\ T_4 \in [8,20],\ T_6 \in [15,50]$
Constraints on Deadlines & Phases
$\phi_5 \geq d_3 + 10 \quad \phi_5 + d_5 \leq 35$
$\phi_5 \geq d_4 + 10 \quad d_4 + d_6 \leq 45$
Equalities
$T_1, T_3, T_7 = T_5\ T_2 = T_4,\ T_8 = T_6$
$d_1, d_2, d_7, d_8 = 0\ \phi_1, \phi_2 = 0$
$\phi_3 = \phi_1 + d_1 + 5\ \phi_4 = \phi_2 + d_2 + 5\ \phi_6 = \phi_4 + d_4 + 5$
$\phi_7 = \phi_5 + d_5 + 5\ \phi_8 = \phi_6 + d_6 + 5$

Table 2. Intermediate Constraints.

4 Solving the Intermediate Constraints

The solution of the intermediate constraints is done in two steps: namely period assignment and phase and deadline assignment. Each of these steps is further carried out in several steps. Figure 4 depicts the procedure for solving the constraints.

The period assignment component accepts the constraints on periods as input and generates task periods with the objective of minimizing processor utilization. This is not an easy problem as it involves nonlinear integer programming due to the harmonicity requirements. To attack this problem, we convert the original optimization problem into a decision problem through the use of *cut-off* utilizations which are provided by the user. Our approach to the period assignment problem, then, is to perform a branch-and-bound search after we reduce the search space of the problem as much as possible. To reduce the search space, we use three pruning algorithms, namely, *Time Granularity Pruning*, *Utilization Pruning*, and *Harmonicity Pruning*.

Once the periods have been assigned, the deadline and phase assignment component takes these results and the constraints on phase and deadlines as input to obtain phase and deadline values. This component is subsequently composed of the *Phase Variable Elimination*, *Deadline Decomposition*, and *Phase Assignment* steps. If any of these steps fails, we return to the period assignment component to obtain a new period assignment. We elaborate on each of these two components and their respective steps in the following two subsections.

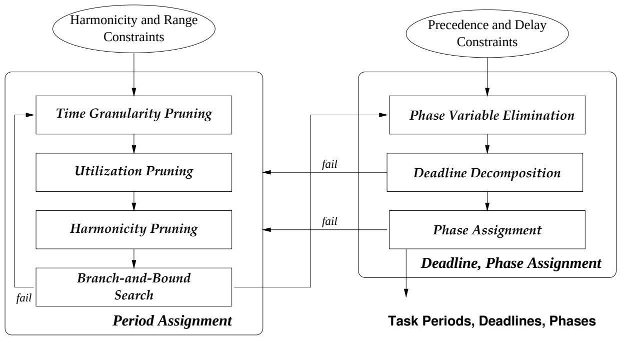

Figure 4. The intermediate constraint solving procedure.

4.1 Period Assignment

As mentioned above, this component consists of the pruning and search steps. These steps are illustrated using the walk-through example. Recall that the reduced set of period constraints only involve T_4, T_5, and T_6. The initial feasible range for the periods is $T_4 \in [8, 20]$, $T_5 \in [9, 20]$, and $T_6 \in [15, 50]$ as shown in Table 2.

1. *Time Granularity Pruning:* Our first pruning takes place by choosing a coarser time granularity. A coarser granularity results in smaller search space, but may also eliminate feasible solutions. Selecting the right value is problem specific, therefore, we begin with a large value and iteratively reduce it if a solution is not found. For our example, we choose a time granularity of 5, resulting in the following feasible ranges for the periods: $T_5 \in \{10, 15, 20\}$, $T_4 \in \{10, 15, 20\}$, and $T_6 \in \{15, 20, 25, 30, 35, 40, 45, 50\}$
2. *Utilization Pruning:* In this step, we use the *cut-off utilization* for each host to tighten the lower bound of the periods. For a set $\mathcal{T}^k$ of tasks allocated on host $\mathcal{P}^k$, the processor utilization U^k must be less than the cut-off utilization U_c^k, i.e., $U^k = \sum_{\tau_i \in \mathcal{T}^k} \frac{e_i}{T_i} \leq U_c^k$. To obtain a lower bound for the period of τ_i, we solve the above by plugging in the largest period values for $\tau_j \in \mathcal{T}^k - \{\tau_i\}$. In our example, we use 0.9 as the cut-off utilization for each host. This results in the following feasible set of values: $T_4 \in \{15, 20\}$, $T_5 \in \{15, 20\}$, and $T_6 \in \{35, 40, 45, 50\}$.
3. *Harmonicity Pruning:* A final pruning method we use is based on the harmonicity relationships. This results in removal of any values from the range of a variable for which we cannot find values for its predecessor or successor tasks in the harmonicity graph. For example, the values 35 and 50 for T_6 may be eliminated, since neither 15 nor 20 (the valid values for T_4) satisfies the harmonicity constraint $T_6 | T_4$. This pruning is repeated until no further reduction is possible.

After the three pruning steps, we are left with a reduced search space on which the search is performed to find a feasible solution. A simple branch-and-bound heuristic[6] is used to control the search. As a result, we obtain the following period assignment.

	T_3	T_4	T_5	T_6	U^1	U^2
Solution	20	20	20	40	0.75	0.825

4.2 Phase and Deadline Assignment

Once the periods are known, we proceed to solve for phase and deadline variables. This process is done in several steps, as outlined below.

(1) *Phase Variable Elimination.* We begin by eliminating the phase variables from the constraint set using Fourier Variable Elimination [4]. Basically, this involves rewriting the constraints as lower and upper bound constraints on a variable to be eliminated, and then combining each lower bound with each upper bound. The elimination of variables is done in a reverse topological sort order of the task

graph. For our example system, we have only one free phase variable, which is eliminated as shown below:

$$\begin{array}{rcl} \phi_5 & \geq & d_3 + 10 \\ \phi_5 & \geq & d_4 + 10 \\ \phi_5 & \leq & 35 - d_5 \\ d_4 + d_6 & \leq & 45 \end{array} \quad \stackrel{\phi_5}{\Longrightarrow} \quad \begin{array}{rcl} d_3 + d_5 & \leq & 25 \\ d_4 + d_5 & \leq & 25 \\ d_4 + d_6 & \leq & 45 \end{array}$$

(2) *Deadline Decomposition.* At this stage, we have a system of linear constraints on the deadlines, which are of the following form:

$$d_{i_1} + d_{i_2} + \cdots + d_{i_{n_i}} \quad \leq \quad D_i$$

Let $\mathcal{C} = C_1, C_2, \ldots, C_n$ denote the set of such constraints, where each constraint C_i reflects the end-to-end constraint on some transaction Γ_i. Consider the periodic task set $\mathcal{T} = \{\tau_i = \langle T_i, e_i, d_i \rangle :: 1 \leq i \leq n\}$, for which the task periods and execution times are known. We use the notion of *critical scaling factor* [10] for this task set as our objective function for solving the constraints. Let $\mathcal{D} = \{d_i :: 1 \leq i \leq n\}$ be any solution satisfying $\mathcal{C}$. Then, we define $\rho^*(\mathcal{D})$, as the largest value of ρ, such that the task set $\mathcal{T}(\rho) = \{\tau_i = \langle T_i, \rho e_i, d_i \rangle :: \ 1 \leq i \leq n\}$. is schedulable. The critical scaling factor $\rho^*(\mathcal{D})$ of a solution thus refers to the capacity to accept $e_i \rho^*(\mathcal{D})$ computation demand for each task in $\mathcal{T}$, without sacrificing schedulability. As with period assignment, we use a cut-off value ρ^*_c, and find any solution $\mathcal{D}$ such that $\rho^*(\mathcal{D}) \geq \rho^*_c$.

We present a heuristic solution strategy for this problem in Section 4.2.1. For now, we just state that the solution obtained from our heuristic solver for the example system is $d_3 = 15$, $d_4 = 8$, $d_5 = 9$, and $d_6 = 36$.

(3) *Phase Assignment.* Finally, when the deadlines have been assigned, we proceed to determine values for task phases. This is done by assigning values to each of the phase variables eliminated in Step 1. The values are assigned in the reverse order of elimination, i.e., in a topological sort order. For each phase variable, we assign the smallest value which satisfies the constraints. For our system, we have only one phase variable ϕ_5, which is assigned the value 25. The values for remaining phase and deadline variables are automatically assigned through the equalities derived in Section 3. The final task set parameters for the example system are shown below.

	e	ϕ	T	D		e	ϕ	T	D
τ_1	0	0	20	0	τ_2	0	0	20	0
τ_3	7	5	20	15	τ_4	8	5	20	8
τ_5	9	25	20	9	τ_6	15	18	40	36
τ_7	0	39	20	0	τ_8	0	59	20	0

4.2.1 Deadline Decomposition

Currently, we adopt an approximate heuristic strategy to solve this problem. Our first approximation involves using static priority scheduling model to check for schedulability. While not optimal, it reduces the search space, and also allows for easier response time determination. Let $\Pi = \{p_i :: 1 \leq i \leq n\}$ be any priority ordering, and let $r_i(\Pi, \rho)$ denote the response time of task τ_i in the task set $\mathcal{T}(\rho)$. Then, the critical scaling factor $\rho^*(\Pi)$ is defined as the largest value of ρ such that $\mathcal{T}(\rho)$ is schedulable under the priority ordering Π. Our second approximation involves using an approximate notion of critical scaling factor, which we refer to as *gain*, and represent as $\hat{\rho}$. The gain of a priority ordering $\hat{\rho}(\Pi)$ is the defined as the largest value of ρ, such that the set of deadlines $d_i = \rho * r_i(\Pi)$ satisfies the constraint set $\mathcal{C}$.

Starting from equal priority for all tasks, we perform the search for a priority ordering through an iterative refinement process. To help in the refinement process, we define per-task gain, denoted as $\hat{\rho}_j(\Pi)$, as follows: Let $\mathcal{C}(j)$ be the subset of constraints in $\mathcal{C}$ which involve the variable d_j. Then $\hat{\rho}_j(\Pi)$ is the largest value of ρ such that the set of deadlines $d_i = \rho * r_i(\Pi)$ satisfies the constraint set $\mathcal{C}(j)$. Clearly, $\hat{\rho}(\Pi) = \min_j \hat{\rho}_j$. In each step, given the priority ordering, we first compute the response times of all tasks, as in [3], assuming that a task is preemptable by all higher and equal priority tasks. Then, based on response times, per task gains are computed. If all gains are above the cut-off threshold, then we stop. Otherwise, the priority of the lowest gain task is elevated above all its equal priority tasks. The procedure fails when the lowest gain task has no equal priority task.

The refinement steps for our walk-through example are illustrated in Table 3. Recall that the constraint set is:

$$d_3 + d_5 \leq 25, \quad d_4 + d_5 \leq 25, \quad \text{and} \quad d_4 + d_6 \leq 45$$

Since a feasible solution has been found, the final deadlines are determined as $d_i = \lfloor \hat{\rho}_i * r_i \rfloor$:

$$d_3 = 15, \quad d_4 = 8, \quad d_5 = 9, \quad \text{and} \quad d_6 = 36$$

Priorities	Response Times	Gains
$p_3 = p_4$ $p_5 = p_6$	$r_3 = 15$ $r_4 = 15$ $r_5 = 24$ $r_6 = 33$	$\hat{\rho}_3 = 0.64$ $\hat{\rho}_4 = 0.64$ $\hat{\rho}_5 = 0.64$ $\hat{\rho}_6 = 0.94$
$p_3 = p_4$ $\mathbf{p_5 < p_6}$	$r_3 = 15$ $r_4 = 15$ $r_5 = 9$ $r_6 = 33$	$\hat{\rho}_3 = 1.04$ $\hat{\rho}_4 = 0.94$ $\hat{\rho}_5 = 1.04$ $\hat{\rho}_6 = 0.94$
$\mathbf{p_4 < p_3}$ $p_5 < p_6$	$r_3 = 15$ $r_4 = 8$ $r_5 = 9$ $r_6 = 33$	$\hat{\rho}_3 = 1.04$ $\hat{\rho}_4 = 1.04$ $\hat{\rho}_5 = 1.04$ $\hat{\rho}_6 = 1.10$

Table 3. Deadline Decomposition

5 Conclusion

We have presented a resource conscious methodology for designing distributed real-time systems. Specifically, we show how a real-time system design specified as a task graph with end-to-end timing constraints can be systematically transformed into a schedulable set of periodic tasks. This methodology enables developers to streamline the end-to-end design of real-time systems by way of a semi-automatic tool-based approach. We believe that a tool developed using the ideas developed in the paper will be very useful for rapid prototyping of designs, and in identifying and eliminating bottlenecks. We have successfully done a case study of our period-assignment algorithm for computerized numeric control [8] in a single processor environment. We hope to extend it for distributed systems using our deadline decomposition strategy.

There exist many directions along which our approach can be extended. First, we have assumed allocation of tasks to processors. We would like to include that in the constraint solving process. Second, we want to extend our design model to incorporate more elaborate task structures, communication mechanisms, and timing constraints. Finally, it would be nice to do some sensitivity analysis; since our method works at design time, the execution times are likely to be crude estimates. Therefore, it is desirable that small changes in execution times do not involve starting from scratch.

References

[1] N. Audsley, A. Burns, and A. Wellings. Data consistency in hard real-time systems. *Informatica*, 19(2):223–234, May 1995.

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

[3] A. Burns. Preemptive priority based scheduling: An appropriate engineering approach. In S. Son, editor, *Principles of Real-Time Systems*. Prentice Hall, 1994.

[4] G. Dantzig and B. Eaves. Fourier-Motzkin Elimination and its Dual. *Journal of Combinatorial Theory (A)*, 14:288–297, 1973.

[5] J. Garcia and M. G. Harbour. Optimized priority assignment for tasks and messages in distributed hard real-time system. In *Proceedings, IEEE Workshop on Parallel and Distributed Real-Time Systems*, 1995.

[6] R. Gerber, S. Hong, and M. Saksena. Guaranteeing real-time requirements with resource-based calibration of periodic processes. *IEEE Transactions on Software Engineering*, 21(7), July 1995.

[7] H. Gomaa. *Software Design Methods for Concurrent and Real-Time Systems*. Addison-Wesley Publishing Company, 1993.

[8] N. Kim, M. Yoo, S. Hong, M. Saksena, C. Choi, and H. Shin. Visual assessment of a real-time system design: A case study of period calibration method. In *Proceedings, IEEE Real-Time Systems Symposium*, 1996.

[9] M. Klein, J. Lehoczky, and R. Rajkumar. Rate-monotonic analysis for real-time industrial computing. *IEEE Computer*, pages 24–33, Jan. 1994.

[10] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 166–171. IEEE Computer Society Press, Dec. 1989.

[11] M. D. Natale and J. Stankovic. Dynamic end-to-end guarantees in distributed real-time systems. In *Proceedings, IEEE Real-Time Systems Symposium*, pages 216–227, 1994.

[12] K. Ramamritham and J. A. Stankovic. Scheduling algorithms and operating systems support for real-time systems. *Proceedings of the IEEE*, 82(1):55–67, January 1994.

[13] D. Seto, J. P. Lehoczky, L. Sha, and K. G. Shin. On task schedulability in real-time control systems. In *Proceedings, IEEE Real-Time Systems Symposium*, 1996.

[14] L. Sha and S. S. Sathaye. A systematic approach to designing distributed real-time systems. *IEEE Computer*, 26(9):68–78, September 1993.

[15] D. Simon, B. Espiau, E. Castillo, and K. Kapellos. Computer aided design of a generic robot controller handling reactivity and real-time control issues. *IEEE Transactions on Control Systems and Technology*, 1(4), Dec. 1993.

[16] J. Sun, R. Bettati, and J. W.-S. Liu. An end-to-end approach to schedule tasks with shared resources in multiprocessor systems. In *Proceedings of the 11th IEEE Workshop on Real-Time Operating Systems and Software*, Seattle, Washington, May 1994.