Submitted September 9, 1994. Revised November 15, 1996 for IEEE Transactions on Software Engineering.

Algorithms for Scheduling Real-Time Tasks with Input Error and End-to-End Deadlines

W. Feng and J. W.-S. Liu
Department of Computer Science
University of Illinois at Urbana-Champaign
1304 W. Springfield Ave.
Urbana, IL 61801
{feng, janeliu}@cs.uiuc.edu

Abstract

This paper describes algorithms for scheduling preemptive, imprecise, composite tasks in real time. Each composite task consists of a chain of component tasks, and each component task is made up of a mandatory part and an optional part. Whenever a component task uses imprecise input, the processing times of its mandatory and optional parts may become larger. The composite tasks are scheduled by a two-level scheduler. At the high level, the composite tasks are scheduled preemptively on one processor, according to an existing algorithm for scheduling simple imprecise tasks. The low-level scheduler then distributes the time budgeted for each composite task across its component tasks so as to minimize the output error of the composite task.

1 Introduction

A hard real-time system contains tasks which must produce logically correct results within certain timing constraints. In a system where the processing times of tasks vary, transient overloads may be unavoidable. A hard real-time system must remain robust and maintain an acceptable level of performance under a transient overload. The imprecise-computation technique [1-5] was introduced as a way to deal with transient overloads. The technique is motivated by the fact that one can often trade off precision for timeliness. It prevents missed deadlines and provides graceful degradation during a transient overload by ensuring that an approximate result of acceptable quality is available whenever the exact result cannot be obtained in time.

The imprecise-computation model used in previous studies [3-15] assumes that the quality of a task's result depends solely on the time spent by the task to produce the result. Specifically, the input of each task is free of error. If a task terminates prematurely, the result produced by it contains an error that is a nondecreasing function of the processing time of the unexecuted portion. While this model adequately characterizes many real-time applications, there are several others that it does not. Examples include video compression, speech recognition, and radar tracking. In these applications, the quality of a result produced by a task depends also on the quality of the input of the task. When the results produced by some tasks are used as inputs by other tasks, the decision on how much of each task to complete by what time in order for the set of dependent tasks to produce a good overall result should not be made by considering each task independently from the others.

In addition, previous studies on imprecise computation focus on the case where the timing constraints of each task are given. However, the timing constraints that can be derived directly from high-level requirements are typically not that of individual tasks, but rather are timing constraints of sets of tasks. We call such timing constraints *end-to-end timing constraints*. The end-to-end timing constraints over each set of tasks must be met. In contrast, the individual tasks in the set do not have any specific timing constraints, other than those imposed by the end-to-end timing constraints. Thus, we have the freedom to advance and postpone the executions of individual tasks. This freedom gives us an added dimension in the tradeoff between result quality and timing requirements.

In this paper, we extend the imprecise-computation model to account for input error as well as the end-to-end nature of the timing constraints. According to this model, tasks which jointly support a function of the system are dependent. The amount of time required to complete a successor task depends on the quality of the result produced by its predecessor task. Each set of dependent tasks forms a composite task. Composite tasks are independent of each other. The ready time and deadline of each composite task are the end-to-end timing constraints of the component tasks in it. We describe in this paper a two-level approach to scheduling composite tasks. At the high level, the scheduler determines the total amount of time budgeted to each composite task in order for all the composite tasks to meet their deadlines. At the low level, the scheduler distributes the time budgeted for each composite task to its component tasks so as to minimize its output error. The high-level scheduler can use one of several existing algorithms for scheduling independent tasks with precise inputs to determine the time budgets of the composite tasks. We describe here several heuristic algorithms which the low-level scheduler can use to distribute time to component tasks.

The remainder of this paper is organized as follows: Section 2 provides the background information needed for Section 3, which describes the extended imprecise-computation model. Section 4 quantifies the effect of input error on the processing time requirements and result quality of dependent component tasks. Section 5 presents a set of heuristic scheduling algorithms which attempt to optimize the overall result quality of each composite task. Section 6 presents the performance of these algorithms. Lastly, Section 7 summarizes our results and presents future work.

2 Background

Our model is built on the imprecise-computation model used in previous studies [3-15] which characterizes the workload on a real-time system as a set of preemptable tasks $\mathbf{T} = \{T_1, T_2, \ldots, T_q\}$. Each task T_i is logically decomposed into a *mandatory part* M_i followed by an *optional part* O_i and has the following rational parameters:

- *ready time* r_i: the time at which task T_i is available for execution,
- *deadline* d_i: the time by which task T_i must produce a result,
- *processing time* p_i: the amount of processor time required to execute the task T_i to completion,
- *mandatory processing time* m_i: the amount of processor time required to execute the mandatory part M_i to completion, and
- *optional processing time* o_i: the amount of processor time required to execute the optional part O_i to completion.

The mandatory part M_i must execute to completion in order to produce an acceptable and usable result. The task T_i meets its deadline if its mandatory part completes by its deadline. The optional part O_i can only execute after the mandatory part M_i completes. The optional part may be terminated before it has completed, if necessary, so that the task and other tasks can meet their deadlines.

Our attention is confined to uniprocessor systems. We let ϕ_i denote the amount of processor time that is assigned to execute the task T_i, according to a given schedule. A scheduling algorithm works correctly if it never schedules a task before its ready time and if the total time ϕ_i it assigns to every task T_i is no less than the mandatory processing time m_i. We let $\sigma_i = \phi_i - m_i$ be the amount of time remaining, after the mandatory part M_i completes, for the execution of the optional part O_i. σ_i may be less than o_i. Hereafter, by a scheduling algorithm , we mean a correct algorithm, and by a schedule, we mean a *valid schedule* in which $m_i \leq \phi_i \leq p_i$. We call a schedule in which every task T_i is assigned m_i or more units of time before its deadline a *feasible schedule*.

When the assigned processor time ϕ_i equals p_i, or equivalently σ_i equals o_i, the task T_i is said to be *precisely* scheduled. Otherwise if $\phi_i < p_i$, the last portion of T_i with processing time $p_i - \phi_i$ is discarded. In a valid schedule, $p_i - \phi_i$, which is equal to $o_i - \sigma_i$, is always equal to or less than o_i. We call $p_i - \phi_i$ the *amount of discarded work* and

$$F_i = \frac{o_i - \sigma_i}{o_i} = \frac{p_i - \phi_i}{o_i} \quad \text{for } \ m_i \leq \phi_i \leq p_i$$

the *fraction of discarded work*[1]. Since ϕ_i is never in the range $[0, m_i)$, the value of F_i in this range is irrelevant; it is assumed to have the value of one for the sake of convenience.

Most existing algorithms [3-5, 7-15] for scheduling tasks with optional parts assume that the tasks are monotone. As a *monotone task* executes longer, the quality of its results improves. Hence, these algorithms seek schedules in which the fraction of discarded work (or some function of this fraction) of each task is as small as possible. Specific performance metrics commonly used by existing algorithms include weighted average of the fractions of discarded work, sum of the fractions of discarded work, maximum fraction of discarded work, and number of discarded optional tasks.

The high-level scheduler described later in this paper makes use of Algorithm G described in [15]. This optimal off-line algorithm finds preemptive schedules of independent tasks in which the maximum fraction of discarded work among all tasks is as small as possible. This algorithm in turn uses the optimal algorithm, developed earlier by Shih et al. [11], for scheduling off-line preemptive tasks with arbitrary ready times and deadlines to minimize the sum of the amounts of discarded work over all tasks. While these algorithms allow tasks to be dependent, the possibility

[1] In previous studies on imprecise computation, E_i was called the error in the result of task T_i when the quality of the result was assumed to be a linear function of the amount of discarded work.

of error propagation across dependent tasks was not considered. Shih et al. [12] has since modified this algorithm to schedule tasks on-line so that the sum of the amounts of discarded work of all tasks accepted by the system is minimized.

The concept of trading off result quality for timeliness has also been independently studied by the artificial intelligence community. Dean and Boddy [16, 17] proposed the use of *anytime algorithms* in the framework of time-dependent planning and decision-making. The execution of an anytime algorithm may be interrupted at any point to return a result whose quality is solely a function of the processing time of the completed portion. Therefore, a task based on the anytime algorithm is an optional task in the imprecise-computation model: The mandatory processing time of such a task is zero.

Zilberstein [18, 19, 20] extended the work of Dean and Boddy by introducing the concept of *conditional performance profiles.* Conditional performance profiles represent result quality as a function of input quality as well as the processing time spent to produce the result. The extended imprecise-computation model described in the next section resembles Zilberstein's model in this way.

Musliner [21] introduced the concept of *any-dimension algorithms* — a general class of iterative algorithms. This concept generalizes the anytime algorithm concept by providing guarantees along dimensions other than time. Like an anytime algorithm, an any-dimension algorithm is an iterative algorithm with a termination condition which halts the iteration when some threshold along that dimension is reached. This aspect is not considered here.

Dey, Kurose, and Towsley [22] introduce the notion of *reward*, which is analogous to *error* in the imprecise-computation model or *quality* in anytime algorithms. They schedule tasks so that tasks receive *increasing reward with increasing service* (IRIS). However, their model ignores the dependency of result quality on input quality.

3 Extensions to the Imprecise-Computation Model

We extend here the imprecise-computation model to characterize applications where inputs provided to tasks may be imprecise. In the extended imprecise-computation model, a workload consists of a set of independent, preemptable composite tasks $\mathbf{T} = \{T^1, T^2, \ldots, T^q\}$. In particular, the input of each composite task is independent of the the output produced by other composite tasks. Each composite task T^j consists of n_j component tasks $T_1^j, T_2^j, \ldots, T_{n_j}^j$. The parameters p^j, m^j, and o^j are the processing time, mandatory processing time, and optional processing time of the composite task T^j, respectively. They are equal to the sums of the corresponding parameters of the component tasks in T^j; that is, $p^j = \sum_{i=1}^{n_j} p_i^j$, $m^j = \sum_{i=1}^{n_j} m_i^j$, and $o^j = \sum_{i=1}^{n_j} o_i^j$, where p_i^j, m_i^j, and o_i^j are the processing time, mandatory processing time, and optional processing time of component task T_i^j.

We focus our attention on the type of composite tasks whose component tasks have linear precedence-constraint graphs. The (end-to-end) ready time r^j of the composite task T^j is the ready time of its first component task T_1^j. T_1^j can begin execution at r^j. T_i^j can begin execution only after T_{i-1}^j completes for $i = 2, 3, \ldots, n_j$. T_i^j is the (immediate) successor of T_{i-1}^j, and the output of T_{i-1}^j is the input of T_i^j. The output of the last component task $T_{n_j}^j$ is the output of the composite task. The (end-to-end) deadline d^j of T^j is the deadline its last component task $T_{n_j}^j$. The ready times and deadlines of the intermediate component tasks $T_2^j, T_3^j, \ldots, T_{n_j-1}^j$ are not specified. To capture the end-to-end nature of the timing constraints, we assign the ready times and deadlines of the component tasks to be the same as that of the composite task. This gives us the maximum flexibility in scheduling the component tasks.

Most of this paper deals with the scheduling of component tasks within a given composite task. When there is no ambiguity, we simplify our notation by dropping the superscript j from the component tasks $T_1^j, T_2^j, \ldots T_n^j$ of the composite task T^j and simply call them tasks $T_1, T_2, \ldots T_n$. We assume that all of the component tasks are monotone.

When a component task T_{i-1} is terminated before it is completed, its output contains an error E_{i-1} which is the input error e_i of its successor task T_i. One effect of the input error e_i is that the mandatory part M_i of T_i may be extended in the sense that it now takes an additional $H_i(e_i)$ units of processor time beyond m_i for T_i to produce an acceptable result. We call the $H_i(e_i)$ additional time units of the mandatory part the *mandatory extension* of T_i, and the concatenation of the mandatory part M_i and the mandatory extension the *extended mandatory part* M_i'. The amount of processor time needed to execute M_i' to completion is given by $m_i' = m_i + H_i(e_i)$. Throughout this paper, we assume that the mandatory extension $H_i(e_i)$ is a monotone nondecreasing function of the input error e_i and $H_i(0) = 0$. Since every component task is monotone, the input error e_i of the task T_i is a monotone nondecreasing function of the fraction of discarded work F_{i-1} of the predecessor task T_{i-1}. Consequently, the mandatory extension of T_i is a monotone nondecreasing function of F_{i-1}. With a slight abuse of notation, we denote this function by $H_i(F_{i-1})$, for $i = 2, 3, \ldots, n$. The mandatory extension of T_i is zero if its predecessor task T_{i-1} is assigned p_i units of time, i.e., $F_{i-1} = 0$, and $H_i(0) = 0$ for all i. We assume that the input error e_1 to the first component task in every composite task is zero, and hence $H_1(e_1) = 0$.

The effect on the processing time of the mandatory part can be illustrated by Newton's root-finding method. Suppose that the processing time of the mandatory part is the time it takes to find a root within 10% of the actual root, and the processing time of the optional part is the time it takes to refine the result of the mandatory part to be within 0.1% of the actual root. Obviously, if the input to Newton's method is poor (i.e., the predecessor component task provides an input value which is far from the actual root), the mandatory part must execute longer in order to meet the 10% threshold. This effect is accounted for by the mandatory extension. The optional part stays the same.

Similar examples of extending the mandatory part occur in image and video processing as well. Suppose that the processing time of the mandatory part is the time it takes to display (on a client workstation) a video frame of acceptable quality, and the processing time of the optional part is the time it takes to further enhance the visual quality of the video frame. If during the transmission of a frame to the client, some packets are lost or corrupted; the client must do some image enhancement, and the mandatory part of displaying the video frame is extended correspondingly in order to bring the quality of the video frame up to acceptable standards.

Another possible effect of input error on a task T_i is that the processing time of the optional part O_i is lengthened. Based on the same argument above, the processing time of the optional part O_i of T_i is extended by $K_i(F_{i-1})$ time units to offset the effects of the input error, where $K_i(x)$ is a monotone nondecreasing function of x and $K_i(0) = 0$. The $K_i(F_{i-1})$ additional time units of the optional part make up the *optional extension*, and the juxtaposition of the optional part O_i and the optional extension is the *extended optional part* O'_i. The amount of processor time needed to execute O'_i to completion is given by $o'_i = o_i + K_i(F_{i-1})$ for $i = 2, 3, \ldots, n$, and $o'_1 = o_1$ because

$K_1(0) = 0$.

Radar tracking is an example of an application where extending the optional part is necessary to compensate for input error. The mandatory part consists of processing the returned radar signal and creating track records by a signal processor. Each track record indicates position and direction of movement of a possible target. The optional part tries to associate these targets with established tracks by a data processor. If the returned radar signal is weak and noisy, the amount of time required to process the signal remains more or less the same (i.e., the processing time of the mandatory part stays the same). However, there may be more false returns — records associated with nonexistent targets. As a result, the data processor must spend additional processing time in associating tracks.

In general, both the mandatory part M_i and the optional part O_i may be extended when a fraction F_{i-1} of the optional part O_{i-1} is discarded and hence the input error e_i is nonzero. Specifically, the mandatory part M_i is extended by $H_i(F_{i-1})$ and the optional part O_i by $K_i(F_{i-1})$. An infinite value of $H_i(F_{i-1})$ means that the input error e_i of T_i has a fatal effect on T_i: T_i can never produce an acceptable result no matter how long it executes. Similarly, if $K_i(F_{i-1})$ is infinite while $H_i(F_{i-1})$ is finite, the effect of input error can never be erased by executing the optional part O_i' longer and longer, although the result produced by T_i is acceptable. Hereafter, we will refer to $H_i(F_{i-1})$ and $K_i(F_{i-1})$ as the *mandatory* and *optional extension functions*, respectively.

4 Effects of Input Error

Again, we consider here composite tasks whose precedence-constraint graphs are linear. For such a composite task, its output error is equal to the output error of the last component task, which, in turn, depends on the output errors of its predecessor component tasks.

Let Φ^j denote the total amount of processor time assigned to a composite task T^j by the high-level scheduler. The heuristic algorithms for distributing the total time Φ^j to the component tasks attempt to keep the fraction of discarded work F_n of the last component task, and hence the output error E_n of this component task and the composite task, as small as possible.

The algorithms presented in subsequent sections are near optimal when the mandatory extension and optional extension of every component task T_i are linear functions of the fraction of discarded work F_{i-1} of its immediate predecessor T_{i-1}. In particular, they assume that the extension functions of T_i are

$$H_i(F_{i-1}) = h_i F_{i-1} \quad \text{for } i = 2, 3, \ldots, n \tag{1}$$

$$K_i(F_{i-1}) = k_i F_{i-1} \quad \text{for } i = 2, 3, \ldots, n \tag{2}$$

where h_i is the *mandatory error-scaling factor* and k_i is the *optional error-scaling factor* of T_i. However, the assumption on the linearity of $H_i(F_{i-1})$ and $K_i(F_{i-1})$ is not valid in general. We will return in Section 5 to discuss how to approximate arbitrary functions $H_i(F_{i-1})$ and $K_i(F_{i-1})$ by linear functions.

Throughout this section, we assume that the mandatory and optional extensions of component tasks $T_2, T_3, \ldots, T_n$ are given by Eqs. (1) and (2). The mandatory and optional extensions of the first component task T_1 of a composite task are zero since the input of a composite task is assumed to be error-free. We first derive the expression for the fraction of discarded work of the last component task, as a function of the amounts of time assigned to the earlier component tasks for the special case where the optional error-scaling factors k_i are zero; that is, $k_i = 0$ for all $i = 1, 2, \ldots, n_j$. We then consider the special case when the mandatory error-scaling factors h_i are zero. The expressions for the fraction of discarded work of T_i in terms of that of its predecessors will provide clues as to how to distribute the amount of time Φ^j that is assigned to the composite task T^j to its component tasks when the mandatory and optional extensions of all component tasks are given by Eqs. (1) and (2).

When the mandatory error-scaling factor h_i is nonzero and the optional error-scaling factor k_i is zero, the amount of time ϕ_i assigned to each component task T_i must be in the range $[m_i + h_i F_{i-1}, m_i + h_i F_{i-1} + o_i]$ so that its extended-mandatory part M_i' can execute to completion. With ϕ_i constrained to this range, the fraction of discarded work of T_i is a linearly decreasing function of ϕ_i, as illustrated by the dotted line in Figure 1 and stated by the following lemma.

Lemma 4.1 *The fraction of discarded work F_i of a component task T_i whose optional error-scaling factor k_i is zero is given by*

$$F_i = 1 - \frac{1}{o_i}(\phi_i - m_i - h_i F_{i-1})$$

where $m_i + h_i F_{i-1} \leq \phi_i \leq m_i + h_i F_{i-1} + o_i$, when it is assigned ϕ_i units of time and the fraction of discarded work of its predecessor is F_{i-1}.

Proof: The lemma follows directly from the property of similar triangles.

$$\frac{F_i - 0}{\phi_i - (m_i + h_i F_{i-1} + o_i)} = \frac{1 - 0}{m_i + h_i F_{i-1} - (m_i + h_i F_{i-1} + o_i)}$$

□

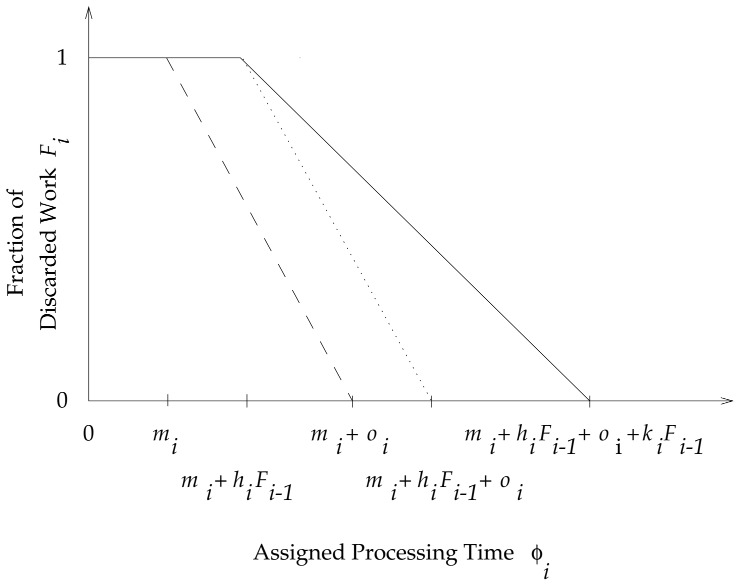

Figure 1: An Extended Imprecise-Computation Model

As an example, if $\phi_i = m_i + h_i F_{i-1}$, then there is only enough time to execute the extended-mandatory part M_i', and the optional part O_i is left unexecuted. As a result, $F_i = 1$. In general, the fraction of discarded work F_n of a composite task consisting of n component tasks is given by the following theorem.

Theorem 4.1 *For a composite task with n component tasks whose optional error-scaling factors are zero, the fraction of discarded work F_n of the last component task T_n is given by*

$$F_n = C_n - a_1\phi_1 - a_2\phi_2 - \cdots - a_n\phi_n \tag{3}$$

where ϕ_i is the amount of time assigned to the component task T_i for $i = 1, 2, \ldots, n$, C_n is the constant

$$\begin{aligned} C_n &= 1 + a_1(m_1) + a_2(m_2 + h_2) + \cdots + a_n(m_n + h_n) \\ &= 1 + \frac{m_n}{o_n} + \frac{h_n}{o_n} C_{n-1} \end{aligned}$$

and the coefficients a_i are given by

$$\begin{aligned} a_1 &= \frac{h_2 h_3 \cdots h_n}{o_1 o_2 \cdots o_n} = \frac{h_2}{o_1} a_2 \\ a_2 &= \frac{h_3 h_4 \cdots h_n}{o_2 o_3 \cdots o_n} = \frac{h_3}{o_2} a_3 \\ &\vdots \\ a_{n-1} &= \frac{h_n}{o_{n-1} o_n} = \frac{h_n}{o_{n-1}} a_n \\ a_n &= \frac{1}{o_n} \end{aligned} \tag{4}$$

Proof: By Lemma 4.1, for a composite task consisting of one component task, its fraction of discarded work is given by

$$\begin{aligned} F_1 &= 1 - \frac{1}{o_1}(\phi_1 - m_1) \\ &= C_1 - \frac{1}{o_1}\phi_1 = C_1 - a_1\phi_1 \end{aligned}$$

For a composite task consisting of a chain of two component tasks, the fraction of discarded work F_2 of T_2 is equal to

$$\begin{aligned} F_2 &= 1 - \frac{1}{o_2}(\phi_2 - m_2 - h_2 F_1) \\ &= 1 - \frac{\phi_2}{o_2} + \frac{m_2}{o_2} + \frac{h_2}{o_2}\left(C_1 - \frac{1}{o_1}\phi_1\right) \\ &= C_2 - \frac{h_2}{o_1 o_2}\phi_1 - \frac{1}{o_2}\phi_2 = C_2 - a_1\phi_1 - a_2\phi_2 \end{aligned}$$

Suppose that the amount of discarded work F_{n-1} for a composite task consisting of a chain of $n-1$ component tasks is

$$\begin{aligned} F_{n-1} &= C_{n-1} - \frac{h_2 h_3 \cdots h_{n-1}}{o_1 o_2 \cdots o_{n-1}}\phi_1 - \frac{h_3 h_4 \cdots h_{n-1}}{o_2 o_3 \cdots o_{n-1}}\phi_2 - \cdots - \frac{1}{o_{n-1}}\phi_{n-1} \\ &= C_{n-1} - a_1\phi_1 - a_2\phi_2 - \cdots - a_{n-1}\phi_{n-1} \end{aligned}$$

Then, the amount of discarded work F_n for a composite task with n component tasks is

$$\begin{aligned} F_n &= 1 - \frac{1}{o_n}\left(\phi_n - m_n - h_n F_{n-1}\right) \\ &= 1 - \frac{\phi_n}{o_n} + \frac{m_n}{o_n} + \frac{h_n}{o_n}\left(C_{n-1} - \frac{h_2 h_3 \cdots h_{n-1}}{o_1 o_2 \cdots o_{n-1}}\phi_1 - \frac{h_3 h_4 \cdots h_{n-1}}{o_2 o_3 \cdots o_{n-1}}\phi_2 - \cdots - \frac{1}{o_{n-1}}\phi_{n-1}\right) \\ &= C_n - \frac{h_2 h_3 \cdots h_n}{o_1 o_2 \cdots o_n}\phi_1 - \frac{h_3 h_4 \cdots h_n}{o_2 o_3 \cdots o_n}\phi_2 - \cdots - \frac{1}{o_n}\phi_n \\ &= C_n - a_1\phi_1 - a_2\phi_2 - \cdots - a_n\phi_n \end{aligned}$$

□

Similarly, when the mandatory error-scaling factor h_i of a component task T_i is zero but the optional error-scaling factor k_i is nonzero, the value for ϕ_i must be in the range $[m_i, m_i + o_i']$, where $o_i' = o_i + k_i F_{i-1}$ is the processing time of the extended-optional part.

Lemma 4.2 *The amount of discarded work F_i for a component task T_i whose mandatory part has an error-scaling factor h_i of zero is given by*

$$F_i = 1 - \frac{\phi_i - m_i}{o_i + k_i F_{i-1}} = 1 - \frac{\sigma_i}{o_i + k_i F_{i-1}}$$

where $m_i \leq \phi_i \leq m_i + o_i + k_i F_{i-1}$, when it is assigned ϕ_i units of time and the fraction of discarded work of its predecessor is F_{i-1}.

Proof: Using Figure 1, the lemma follows directly from the property of similar triangles.

$$\frac{F_i - 0}{\phi_i - (m_i + o_i + k_i F_{i-1})} = \frac{1 - 0}{m_i - (m_i + o_i + k_i F_{i-1})}$$

□

Theorem 4.2 *For a composite task with two or more component tasks whose mandatory error-scaling factors h_i are zero, the fraction of discarded work F_i of component task T_i is given by*

$$F_i = 1 - \frac{\sigma_i\,(o_{i-1} + k_{i-1}F_{i-2})}{(o_i + k_i)\,(o_{i-1} + k_{i-1}F_{i-2}) - k_i\,\sigma_{i-1}} \tag{5}$$

where σ_{i-1} and σ_i are the amounts of time assigned to the optional parts of T_{i-1} and T_i, respectively, and F_{i-2} is the fraction of discarded work of T_{i-2}.

Proof: From Lemma 4.2,

$$F_i = 1 - \frac{\sigma_i}{o_i + k_i F_{i-1}} = 1 - \frac{\sigma_i}{o_i + k_i \left[1 - \sigma_{i-1}/(o_{i-1} + k_{i-1} F_{i-2})\right]}$$

□

5 Heuristic Scheduling Algorithms

We now describe a high-level algorithm for scheduling composite tasks and determining the total amount of time assigned to each task. We then focus on low-level algorithms for distributing the time assigned to each composite task among its component tasks. These algorithms make use of the expressions derived in the previous section to decide how the total amount of time assigned to each composite task should be distributed among its component tasks so as to minimize the output error E_n for the composite task. Because the low-level algorithms compute the amounts of time given to the component tasks based on their error-scaling factors, we also present a method for extracting the error-scaling factors.

High-Level Scheduling of Composite Tasks

Since the composite tasks are independent, the quality of the result produced by a composite task is independent of the amounts of time assigned to other composite tasks. Therefore, the problem of scheduling a set of preemptable composite tasks reduces to the problem of scheduling a set of independent preemptable tasks.

Figure 2 gives a pseudo-code description of an algorithm, called S-COMPOSITE, for scheduling composite tasks. (In this figure, we use h^j to denote the sum $\sum_{i=1}^{n_j} h_i^j$; it is the maximum extended-mandatory processing time.) This algorithm makes use of a modified version of the earliest-deadline-first algorithm (M-EDF), described in [11], and Algorithm G, described in [15]. The M-EDF algorithm treats every composite task as if it were entirely optional and schedules the composite tasks on an earliest-deadline-first basis. It never schedules any composite task after its deadline. In other words, every composite task is terminated at its deadline if it is not completed. It has been shown in [11] that the M-EDF algorithm minimizes the total processing time of the discarded portions of all tasks. Algorithm G distributes the total available processor time as evenly as possible among the tasks. Specifically, for a given set of ready times and deadlines, this algorithm makes the fraction of discarded work of the composite task (i.e., the ratio of the processing time of the unexecuted portion of each composite task to the total optional processing time of all the component tasks in it) as equal as possible to that of the other composite tasks.

Algorithm S-COMPOSITE has four steps. In Step 1, it tries to schedule every composite task T^j precisely. If this step succeeds in finding a precise schedule, the output errors of all the composite tasks are zero, and each component task T_i^j is assigned $m_i^j + o_i^j$ units of time. If Step 1 fails to find a precise schedule of $\mathbf{T}$, no precise schedule of $\mathbf{T}$ exists [11]. Step 2 then tries to precisely schedule every composite task whose optional processing time o^j is small compared to its maximum extended-mandatory processing time. It does so in Step 2(a) by reducing the amounts of time assigned to tasks which have relatively large optional processing times. If, in Step 2(b), the M-EDF algorithm finds a precise schedule with this reduction, the composite tasks with small optional processing times are scheduled precisely and have zero output errors. Each composite task whose optional processing times o^j is larger than its maximum extended-mandatory processing time m'^j is assigned a sufficient amount of time to ensure the feasible distribution of its time among its component tasks.

On the other hand, if Step 2(b) fails, we do not know whether we can feasibly schedule $\mathbf{T}$ until we distribute the time Φ^j assigned to each composite task to its component tasks. Step 3 is carried out and finds Φ^j in such a way that the processing times of the unexecuted portions of all composite tasks, measured in terms of fractions of their processing times, are as equal as possible under the constraints imposed by the ready times and deadlines of the composite tasks. Step 4 is then invoked to distribute this time to the component tasks.

For example, suppose that there are two composite tasks with the following parameters:

Task	m	o	m'	r	d
T^1	15.0	14.0	26.4	0.0	28.5
T^2	45.0	42.0	88.0	27.0	112.0

Figure 3 shows the three schedules of the composite tasks produced in Steps 1, 2, and 3 of Algorithm S-COMPOSITE. Because the schedules produced in Step 1 and Step 2 are not precise, Step 3 is carried out, and the fractions of unexecuted optional portions of both tasks are equal to 1/14. The

Input:

(a) The ready time r^j and deadline d^j of every composite task T^j in $\mathbf{T}$.

(b) The total processing time $p^j = m^j + o^j$ and the maximum extended-mandatory processing time $m'^j = m^j + h^j$.

(c) The parameters m_i^j, o_i^j, h_i^j, and k_i^j of every component task in every composite task in $\mathbf{T} = \{T^1, T^2, \ldots, T^n\}$

Output:

A feasible schedule of $\mathbf{T}$ or a feasible schedule of a subset of $\mathbf{T}$ and the list of composite tasks that cannot be feasibly scheduled.

1. (a) For each composite task T^j, $j = 1, 2, \ldots, q$,
   set the processing time P^j of T^j to p^j.
   (b) Use the M-EDF algorithm to find a schedule of $\{T^1, T^2, \ldots, T^q\}$.
   If the resultant schedule is a precise schedule then
   The output error of every composite task is 0.
   Done.

2. (a) For $j = 1, 2, \ldots, q$, if $P^j > m'^j$
   Then set $P^j = m'^j$.
   (b) Use the M-EDF algorithm to find a schedule of $\{T^1, T^2, \ldots, T^q\}$.
   If the resultant schedule is a precise schedule then
   Set $\Phi^j = P^j$ for $j = 1, 2, \ldots, q$. Go to Step 4.

3. Use Algorithm G to schedule $\mathbf{T}$.
   $\Phi^j =$ the total amount of time assigned to T^j according to the resultant schedule for each composite task in $\mathbf{T}$.

4. (a) For every j, use one of the distribution algorithms to distribute the time Φ^j assigned to T^j among its component tasks.
   (b) Report all infeasible composite tasks and the processing time distribution of all feasible composite tasks.

Figure 2: Algorithm S-COMPOSITE

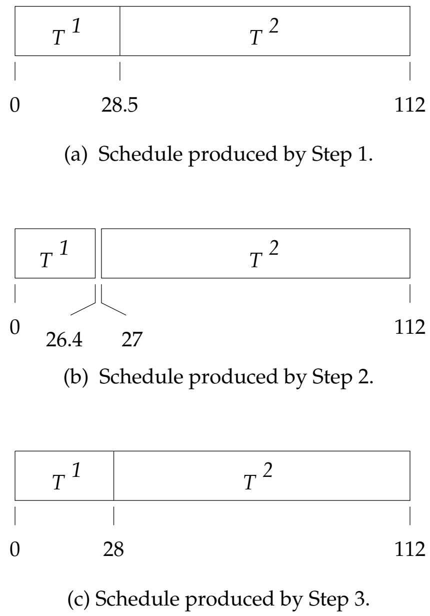

Figure 3: An Example to Illustrate Algorithm S-COMPOSITE

times assigned to these tasks are $\Phi^1 = 28$ and $\Phi^2 = 84$. The processing times of the unexecuted portions of T^1 and T^2 are 1 and 3, respectively.

Later, we will expand this example and show that the algorithm used in Step 4 can find a way to distribute the 28 units of time assigned to T^1 such that the output error is zero and the total time used by its component tasks is less than 28. The results produced by Step 4 include the amount of unused time for each feasibly scheduled composite task and the additional amount of time required for each infeasibly scheduled composite task. We can improve the chance for Algorithm S-COMPOSITE to find a feasible schedule of $\mathbf{T}$ when it fails by carrying out Step 3 and Step 4 again. In the second iteration, the amount of time used by each composite task that is feasibly scheduled in Step 4 in the first iteration is considered to be mandatory. Similarly, the amount of time required by each composite task that is not feasibly scheduled in Step 4 is used as the mandatory processing time of the task in the second iteration.

The time complexity of Steps 1 and 2 is $O(q \; ln \, q)$. The time complexity of Step 3 is $O(q^3)$. The algorithms for distributing time to the component tasks have time complexity $O(n)$. Hence the time complexity of Algorithm S-COMPOSITE is $O(nq + q^3)$.

Low-Level Scheduling to Distribute Processor Time

The Φ units of time assigned to a composite task T must be distributed to its n component tasks in such a way that the following constraints are satisfied.

$$\Phi = \phi_1 + \phi_2 + \cdots + \phi_n \tag{6}$$

$$m_i + h_i F_{i-1} \leq \phi_i \leq m_i + h_i F_{i-1} + o_i + k_i F_{i-1} \tag{7}$$

We present here five algorithms for this distribution. (Since it is no longer necessary for us to keep track of different composite tasks, we again drop the superscript to simplify our notation.)

Algorithms DIST-M, DIST-M$^+$, and DIST-M$^+$-ITERATIVE make decisions on the amount of time given to component tasks based primarily on their mandatory error-scaling factors. They are more suited when the processing times of the mandatory extensions of the component tasks are large compared to the processing times of the optional extensions. In contrast, Algorithms DIST-O and DIST-O$^+$ take into account the effect of the optional error-scaling factors on the output error of the composite task. They should perform better than DIST-M, DIST-M+, and DIST-M+-ITERATIVE when the processing times of the optional extensions are large in comparison to the processing times of the mandatory extensions.

Algorithm DIST-M

Figure 4 gives the pseudo-code description of Algorithm DIST-M, an algorithm which *dist*ributes processor time to component tasks based solely on the *m*andatory error-scaling factor; Theorem 4.1 provides the rationale of this algorithm. According to this theorem, when the optional error-scaling factors of all the component tasks are zero (i.e., $k_i = 0$ for $i = 1, 2, \ldots, n$), the fraction of discarded work F_n is a linear function of the times ϕ_i's assigned to its component tasks. Since the constraints (6) and (7) on the ϕ_i's are also linear, the problem of finding an assignment $\{\phi_i\}$ (i.e., the set of times assigned to component tasks) to minimize F_n, and hence the output error E_n of the composite task, is a linear-programming problem in this special case. We can use a linear-programming package to find $\{\phi_i\}$ if the scheduling is done off-line. However, when $k_i \neq 0$, the assignment $\{\phi_i\}$ thus found is not optimal and, as we will see later, may not even be feasible. Algorithm DIST-M and its variances offer better alternatives with lower scheduling overhead because its time complexity is $O(n)$.

Input:

(a) The parameters m_i, o_i, h_i, and k_i for all n component tasks

(b) The total time Φ assigned to the composite task.

Output:

A feasible assignment $\{\phi_i\}$ of time to the component tasks and the amount of unused time U, or a report on the failure to find a feasible schedule and the additional time required.

1. If $\Phi \geq \sum_{i=1}^{n}(m_i + o_i)$ then
   Schedule component tasks precisely by setting
   $\phi_i = m_i + o_i$ for $i = 1, 2, 3, \ldots, n$ and $E_n = 0$.
   Done.

2. Set $\phi_1 = m_1$ and $\phi_i = m_i + h_i$ for $i = 2, 3, \ldots, n-1$.
   If $\Phi - \sum_{i=1}^{n-1} \phi_i \geq m_n + h_n + o_n + k_n$ then
   Set $\phi_n = m_n + h_n + o_n + k_n$.
   Report $U = \Phi - \sum_{i=1}^{n} \phi_i$.
   Done.

3. Set $F_0 = 0$ and $F_{i-1} = 1$ for $i = 2, 3, \ldots, n$.
   Compute a_i's from h_i's and o_i's for $i = 1, 2, \ldots, n$, according to Eq. (4).
   Sort a_i's in nonincreasing order and put them in the list L.
   While L is not empty do
   x = index of the largest a_i in L
   If T_{x+1} is marked and $x \neq n$ then
   $\phi_x = m_x + h_x F_{x-1}$
   Else $\phi_x = m_x + h_x F_{x-1} + o_x + k_x F_{x-1}$
   Mark T_x and remove a_x from L.

4. Compute the amount of unused time $U = \Phi - \sum_{i=1}^{n} \phi_i$.
   If $U < 0$, then set
   $\phi_1 = m_1$, $\phi_i = m_i + h_i$ for $i = 2, 3, \ldots, n-1$
   $\phi_n = \Phi - \sum_{i=1}^{n-1} \phi_i$
   If $\phi_n < m_n + h_n$ then
   Report that the algorithm fails and that the amount of additional time required is $min(m_n + h_n - \phi_n, -U)$.
   Else report unused amount of time U and $\{\phi_i\}$.

Figure 4: Algorithm DIST-M

More specifically, Algorithm DIST-M has four steps. The first two steps try to find a feasible assignment in which T_n is scheduled to yield a zero output error. If these two steps fail, Step 3 tries to find a good assignment with a small fraction of discarded work F_n. To see the rationale behind this step, we assume for the moment that $k_i = 0$ for all i. From the expression of F_n in Theorem 4.1, we see that this fraction is minimized by choosing the values of ϕ_i in the following way: the larger the coefficient a_i in the sum given by Eq. (3), the larger the value of ϕ_i. Unfortunately, because of constraints (6) and (7), this choice of ϕ_i is not always possible. Because of these constraints and the fact that the k_i's are not zero, Step 3, and hence Algorithm DIST-M, is not optimal.

Step 3 tries to find an assignment which yields a small fraction of discarded work F_n by making locally optimal or near-optimal decisions as follows. It begins by making the value of ϕ_x, which has the largest coefficient a_x among all a_i's in Eq. (3), as large as possible. From Eq. (4), we see that $a_x > a_{x+1}$ means $h_{x+1} > o_x$. In other words, the processing time of the optional part of T_x is smaller than the processing time of the maximum mandatory extension of T_{x+1} when the entire optional part of T_x is left unexecuted. Therefore, ϕ_x is made sufficiently large so that T_x can execute to completion. Similarly, because $a_x > a_{x-1}$, and hence $h_x < o_{x-1}$, less time is required to execute the maximum mandatory extension of T_x than the optional portion of T_{x-1}. Therefore, ϕ_{x-1} is chosen so only the extended-mandatory portion of T_{x-1} completes. This process is repeated until the values of all ϕ_i's are chosen.

Step 4 checks whether the assignment $\{\phi_i\}$ produced in Step 3 is feasible for the total time Φ assigned to the composite task. If the total time required by $\{\phi_i\}$ exceeds Φ, then another attempt to find a feasible assignment is made to produce a schedule with the minimum allowable output error. The algorithm fails if this assignment is also infeasible. As a result of Step 4, the amount of time not used by a feasible assignment $\{\phi_i\}$ or the additional amount of time needed for a feasible assignment is reported. The high-level scheduler can make use of this information to reallocate time among composite tasks, e.g., by giving the time not required by one composite task to one(s) that requires more time.

As an illustrative example, we consider the composite task T^1 in Figure 3 which consists of a chain of four component tasks: T_1, T_2, T_3, and T_4. (We again suppress the superscript.) The parameters of the component tasks are

Task	m	h	o	k
T_1	6.4	0.4	5.0	0.0
T_2	4.0	4.0	2.0	0.0
T_3	1.0	5.0	3.0	0.0
T_4	4.0	2.0	4.0	0.0

The total amount of time assigned to T^1 is 28. T^1 cannot be precisely scheduled and executed because $\sum_{i=1}^{n}(m_i + o_i) = 29$. Hence, Step 1 of Algorithm DIST-M fails to find an assignment that yields zero output error. Step 2 assigns $\phi_1 = 6.4$, $\phi_2 = 8.0$, and $\phi_3 = 6.0$ units of time to T_1, T_2, and T_3, respectively. The remaining time $\Phi - \sum_{i=1}^{3}\phi_i = 7.6$ is insufficient for T_4 to execute to completion. Consequently, Step 3 is carried out. Because $a_4 = \frac{1}{4}$, $a_3 = \frac{2}{3} \times a_4 = \frac{1}{6}$, $a_2 = \frac{5}{2} \times a_3 = \frac{5}{12}$, and $a_1 = \frac{4}{5} \times a_2 = \frac{1}{3}$, we have $a_2 > a_1 > a_4 > a_3$. In the first iteration, $\phi_2 = m_2 + h_2 + o_2 = 10.0$ is set, and T_2 is marked. The fraction of discarded work F_1 of T_1 is 1, and the fraction of discarded work F_2 of T_2 is 0. In the second iteration, a_1 is the largest entry. Because T_2 is marked and $F_1 = 1$, ϕ_1 is set to $m_1 = 6.4$. In the third iteration, a_4 is the largest, and thus ϕ_4 is chosen to be $m_4 + h_4 + o_4 = 10$. Because $F_2 = 0$ and $F_3 = 1$ in the fourth iteration, the time required by T_3 is $m_3 = 1$. Since $\sum_{i=1}^{4}\phi_i = 6.4 + 10.0 + 1.0 + 10.0 = 27.4$, the amount of unused time is 0.6 units, and the fraction of discarded work F_4 of T_4 is zero. Consequently, the output error is zero.

Extensions of Algorithm DIST-M

While Algorithm DIST-M strictly follows Theorem 4.1 when making scheduling decisions, Algorithm DIST-M$^+$ only uses it as a guide. The conceptual difference between the two algorithms is in Step 3. Unlike Algorithm DIST-M, Algorithm DIST-M$^+$ does not make its scheduling decisions based solely on the a_i's. The DIST-M$^+$ algorithm goes one step further by checking to see if the total execution time of the pair of tasks T_i and T_{i+1} is shortened or lengthened by giving the task T_i more processing time. If it is shortened, then task T_i is assigned more time; otherwise, it is only allocated enough time to execute the extended-mandatory part. Steps 1, 2, and 4 of this algorithm are the same as the corresponding steps of Algorithm DIST-M; its Step 3 is described in Figure 5. We note that the purpose of "marking" in this algorithm is different from that in Algorithm DIST-M. Here, a marked task is a task whose fraction of discarded work is zero.

3. Set $F_0 = 0$ and $F_{i-1} = 1$ for $i = 2, 3, \ldots, n$.
    Compute a_i's from h_i's and o_i's for $i = 1, 2, \ldots, n$,
        according to Eq. (4).
    Sort a_i's in nonincreasing order and put them in the
        list L.
    While L is not empty do
        x = index of the largest a_i in L
        If T_{x+1} is marked and $x \neq n$ then
            If $o'_x > h_{x+1}F_x + k_{x+1}F_x$ then
                $\phi_x = m_x + h_x F_{x-1}$
                $\phi_{x+1} = m_{x+1} + h_{x+1}F_x + o_{x+1} + k_{x+1}F_x$
            Else
                $\phi_x = m_x + h_x F_{x-1} + o_x + k_x F_{x-1}$
                $\phi_{x+1} = m_{x+1} + o_{x+1}$
                Mark T_x
        Else /* T_{x+1} is unmarked */
            If $o'_x > h_{x+1}F_x$ then
                $\phi_x = m_x + h_x F_{x-1}$
                $\phi_{x+1} = m_{x+1} + h_{x+1}F_x$
            Else
                $\phi_x = m_x + h_x F_{x-1} + o_x + k_x F_{x-1}$
                $\phi_{x+1} = m_{x+1}$
                Mark T_x
        Remove a_x from L.

Figure 5: Step 3 of Algorithm DIST-M$^+$

The amounts of time required to complete the component tasks may change during the execution of Algorithm DIST-M$^+$. If Algorithm DIST-M$^+$ is applied iteratively, it will produce a schedule as good as or better than a schedule produced by a single pass of Algorithm DIST-M$^+$. Therefore, Algorithm DIST-M$^+$-ITERATIVE repeatedly applies Algorithm DIST-M$^+$ until there are no more changes to the schedule.

Algorithm DIST-O

When the processing times of the mandatory extensions are small compared to the processing times of their optional extensions, we expect that DIST-M and its variances are unlikely to perform well. Algorithm DIST-O offers an alternative. The algorithm consists of three steps. Its first two steps are identical to the first two steps of Algorithm DIST-M; its Step 3 is described in Figure 6.

Step 3 is carried out when Step 2 fails. In other words, after $m_i + h_i$ units of time have been assigned to each of the earlier component tasks T_i, the remaining time $\Phi - \sum_{i=1}^{n-1}(m_i + h_i)$ is insufficient for the last component task T_n to complete. The decision that must be made is whether to assign this time to T_n or an earlier component task or to divide the time between the component tasks in some manner.

After Step 2,
    $\phi_1 = m_1$
    $\phi_i = m_i + h_i$ for $i = 2, 3, \ldots, n-1$

Step 3
    Set $\phi_n = \Phi - \sum_{i=1}^{n-1} \phi_i$.
    If $\phi_n < m_n + h_n$ then
        Report failure to find a feasible assignment and that
            the additional time required is $m_n + h_n - \phi_n$.
        Done.
    Else
        $y = \phi_n - m_n - h_n$
        If $y > o'_n o'_{n-1}/k_n$ then
            $\sigma_{n-1} = min(o_{n-1} + k_{n-1}, y)$
            $\phi_{n-1} = \phi_{n-1} + \sigma_{n-1}$
            $\phi_n = \phi_n - \sigma_{n-1}$
            Report $\{\phi_i\}$. Done.

Figure 6: Step 3 of Algorithm DIST-O

Algorithm DIST-O is based on the expression of output error given by Eq. (5). Like Algorithm DIST-M, the decision made by Algorithm DIST-O is only locally optimal. To understand the reasoning behind the choices made by Step 3, let $y = \Phi - \sum_{i=1}^{n}(m_i + h_i)$ and $y > 0$. We suppose that the y units of time is divided evenly between T_n and T_{n-1}. According to Eq. (5), the fraction of discarded work F_n is given by

$$F_n \quad = \quad 1 - y/(2[o'_n - \frac{k_n y}{2o'_{n-1}}])$$

where again $o'_n = o_n + k_n$ and $o'_{n-1} = o_{n-1} + k_{n-1}$. We now ask whether F_n can be reduced by assigning $y + \delta$ units of time to T_n for $\delta > 0$ at the expense of T_{n-1}. This question is equivalent to whether the inequality

$$(\frac{y}{2} + \delta)/(o'_n - \frac{k_n(y/2 - \delta)}{o'_{n-1}}) \quad > \quad \frac{y}{2}/(o'_{n-1} - \frac{k_n y}{2o'_{n-1}})$$

is true. A simple algebraic manipulation shows that this inequality holds if

$$y \quad < \quad \frac{o'_n\, o'_{n-1}}{k_n}$$

In this case, it is better to give all the remaining time y to T_n. Otherwise, we give more time to the earlier component task T_{n-1}.

Algorithm DIST-O$^+$

The decision made in Algorithm DIST-O is good only when the processing times of the maximum mandatory extensions of the component tasks are fairly small. In general, we may be able to reduce the output error by distributing the available time to earlier component tasks. Algorithm DIST-O$^+$, an enhanced version of Algorithm DIST-O, serves this purpose. This algorithm tries to divide the processing time across all the component tasks; the algorithm is identical to Algorithm DIST-M except that the heuristic guide a_i in Step 3 is calculated differently.

The choice of a_i is motivated by the expression in Eq. (5). According to this equation, when the mandatory error-scaling factors of all the component tasks are zero (i.e., $h_i = 0$ for $i = 1, 2, \ldots, n$), a component task whose optional error-scaling factor and optional processing time are smaller than those of the successor task should be given more time than the successor task when the fraction of discarded work of its predecessor is sufficiently small. Otherwise, the successor task should be given more time. Consequently, we choose a_i to be $o_{i+1}k_{i+1}/o_i k_i$ for $i = 1, 2, \ldots, n$.

Extracting Error-Scaling Factors

The low-level algorithms for distributing time to component tasks require as input parameters the error-scaling factors of the component tasks. These algorithms cannot be applied directly to component tasks whose extension functions are not in the linear form given by Eqs. (1) and (2). We now describe how to transform a given set of component tasks with arbitrary mandatory and optional extensions to another set with linear mandatory and optional extensions and thus find the error-scaling factors.

Without loss of generality, suppose that the given mandatory and optional extensions (denoted by $H_i^g(F_{i-1})$ and $K_i^g(F_{i-1})$, respectively) of a component task T_i are as shown by the dotted curves in Figures 7(a) and (b), respectively. We approximate each given extension function by two straight-line segments; both segments lie entirely above each extension function. These straight-line segments give an upper bound of the values of the extension function at all values of F_{i-1}. A valid, feasible schedule based on such straight-line approximations of the extension functions is surely a valid, feasible schedule for the given extension functions.

Specifically, one of the two straight lines is the vertical one at f_m (or f_o), where $0 < f_m \leq 1$ (or $0 < f_o \leq 1$). We call $f_i = min(f_m, f_o)$, the *discard threshold* of task T_i. The task T_i can never produce a precise result when more than f_i fraction of the optional task O_{i-1} of T_{i-1} is left unexecuted. We want to ensure that this condition never occurs for any component task. For this reason, if $f_i < 1$, we increase the mandatory processing time of T_{i-1} by $(1-f_i)\, o_{i-1}^g$ and decrease the optional processing time of T_{i-1} by $(1-f_i)\, o_{i-1}^g$, where o_{i-1}^g denotes the given optional processing time of T_{i-1}. In other words, we transform the predecessor task T_{i-1} into one whose mandatory and optional processing times are given by

$$\begin{aligned} m_{i-1} &= m_{i-1}^g + (1-f_i)\, o_{i-1}^g \\ o_{i-1} &= o_{i-1}^g - (1-f_i)\, o_{i-1}^g \end{aligned}$$

where m_{i-1}^g is the given mandatory processing time of T_{i-1}. Our algorithms work with the transformed parameters m_{i-1} and o_{i-1}, not the corresponding given parameters.

The allowed range of the fraction of discarded work F_{i-1}, based on the computed value o_{i-1} is $[0, 1]$. Adopting the straight-line segments with finite slopes in this range, we have the following

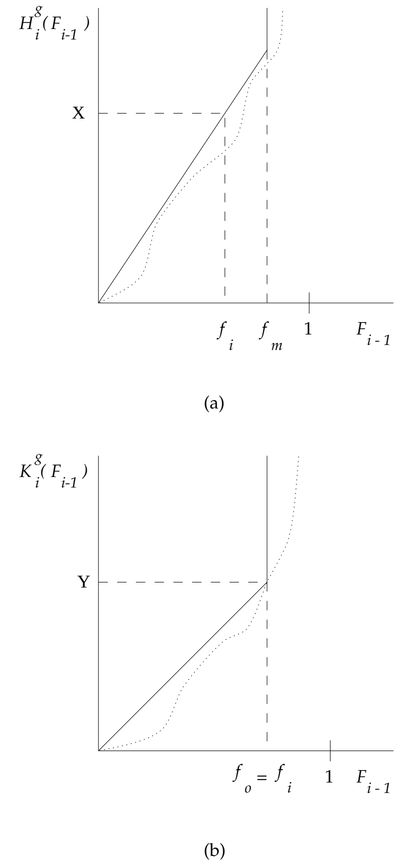

Figure 7: Extension Functions

approximate extension functions:

$$
\begin{aligned}
H_i(F_i) &= XF_{i-1} \\
K_i(F_i) &= YF_{i-1}
\end{aligned}
$$

where X and Y are the values defined in Figure 7. Hence, $h_i = X$ and $k_i = Y$. Hereafter, we assume that the characteristics of all component tasks are first analyzed, and then the parameters m_i, o_i, h_i, and k_i of every component task T_i in every composite task are derived from the given mandatory and optional processing times and extension functions in the manner described above.

6 Performance

We ran a suite of simulation experiments to evaluate the performance of Algorithms DIST-M, DIST-M$^+$, DIST-M$^+$-ITERATIVE, DIST-O, and DIST-O$^+$. For convenience, we call the five algorithms collectively as DIST-* algorithms. In each experiment, we randomly generated a composite task and each of its component tasks. For each component task, the mandatory and optional processing times (i.e., m_i and o_i) and the mandatory and optional error-scaling factors (i.e., h_i and k_i) were randomly chosen from their respective probability distributions; we isolated the effect of these parameters by keeping all other parameters fixed. (For example, the number of component tasks in a composite task was fixed at 8.) For each composite task, the parameters m_i, h_i, o_i, and k_i of each component task T_i were chosen from either uniform distributions or bimodal distributions. We then applied the five algorithms on each composite task and found the fractions of discarded work of the last component task produced by them. For a given algorithm, the lower the fraction of discarded work, the better the performance.

6.1 Uniform Distribution

Table 1 shows the performance of the DIST-* algorithms found in four representative experiments where the parameters m_i, h_i, o_i, and k_i of each T_i were all chosen from uniform distributions. Each column corresponds to a composite task whose component-task parameters are selected from either the uniform distribution in the range $[0, 10)$ or the uniform distribution in the range $[0, 100)$. We call the distribution with the range $[0, 10)$ to be the "small" uniform distribution. So, when the parameter h_i of all component tasks are chosen from the "small" distribution and m_i, o_i, and k_i from the larger distribution, namely the one with range [0,100), we label the appropriate column with "small h" as exemplified by the second column of each experiment in Table 1.

The numerical entries in Table 1 are either in roman or italic font. The first two steps of all five DIST-* algorithms are the same; these steps try to find a feasible scheduling assignment which yields zero output error. When either Step 1 or 2 succeed, the results are listed in roman font. If either of these steps fail, then the succeeding steps distinguish the individual DIST-* algorithms. The results for the composite tasks which benefit from these distinguishing steps are in italics. If none of the steps result in a feasible schedule, then the composite task is said to be unschedulable (uns).

When the expected values of the h_i parameters are small, the five algorithms perform almost identically. Small h_i parameters imply that compensating for the input error of a component task can be *much* cheaper timewise than scheduling its predecessor tasks precisely. As a result, the best

Experiment 1

Algorithm	small														
DIST-	m,h,o,k	h	h,k	h,o	h,o,k	k	o	o,k	m	m,h	m,h,k	m,h,o	m,k	m,o	m,o,k
M	.991	.420	.489	.874	.361	*.161*	.000	.000	uns	.981	.980	.883	*.338*	.000	.000
M$^+$	*.850*	.420	.489	.874	*.211*	*.091*	.000	.000	*.963*	.981	.980	.883	*.268*	.000	.000
M$^+$-ITER	*.850*	.420	.489	.874	*.211*	*.091*	.000	.000	*.963*	.981	.980	.883	*.268*	.000	.000
O	.991	.420	.489	.874	.361	uns	.000	.000	uns	.981	.980	.883	uns	.000	.000
O$^+$	.991	.420	.489	.874	.361	uns	.000	.000	uns	.981	.980	.883	uns	.000	.000

Experiment 2

Algorithm	small														
DIST-	m,h,o,k	h	h,k	h,o	h,o,k	k	o	o,k	m	m,h	m,h,k	m,h,o	m,k	m,o	m,o,k
M	*.144*	.898	.882	.922	*.834*	*.085*	uns	uns	*.026*	.873	.853	.943	*.080*	uns	uns
M$^+$	*.055*	.898	.882	.922	*.735*	*.085*	uns	uns	*.000*	.873	.853	.943	*.080*	uns	uns
M$^+$-ITER	*.055*	.898	.882	.922	*.735*	*.085*	uns	uns	*.000*	.873	.853	.943	*.080*	uns	uns
O	uns	.898	.882	.922	.924	uns	uns	uns	uns	.873	.853	.943	uns	uns	uns
O$^+$	uns	.898	.882	.922	.924	uns	uns	uns	uns	.873	.853	.943	uns	uns	uns

Experiment 3

Algorithm	small														
DIST-	m,h,o,k	h	h,k	h,o	h,o,k	k	o	o,k	m	m,h	m,h,k	m,h,o	m,k	m,o	m,o,k
M	.572	.968	.677	.121	.968	*.914*	.000	.000	uns	.905	.883	.681	*.872*	.000	.000
M$^+$	*.394*	.968	.677	.121	.968	*.914*	.000	.000	*.876*	.905	.883	.681	*.872*	.000	.000
M$^+$-ITER	*.394*	.968	.677	.121	.968	*.914*	.000	.000	*.876*	.905	.883	.681	*.872*	.000	.000
O	.572	.968	.677	.121	.968	uns	.000	.000	uns	.905	.883	.681	*.872*	.000	.000
O$^+$	.572	.968	.677	.121	.968	uns	.000	.000	uns	.905	.883	.681	*.872*	.000	.000

Experiment 4

Algorithm	small														
DIST-	m,h,o,k	h	h,k	h,o	h,o,k	k	o	o,k	m	m,h	m,h,k	m,h,o	m,k	m,o	m,o,k
M	*.208*	.985	.139	.912	*.065*	*.878*	uns	uns	*.018*	.886	.742	.967	*.902*	uns	uns
M$^+$	*.208*	.985	.139	.912	*.170*	*.878*	uns	uns	*.018*	.886	.742	.967	*.902*	uns	uns
M$^+$-ITER	*.208*	.985	.139	.912	*.170*	*.878*	uns	uns	*.018*	.886	.742	.967	*.902*	uns	uns
O	.924	.985	.139	.912	.742	uns	uns	uns	.734	.886	.742	.967	uns	uns	uns
O$^+$	.924	.985	.139	.912	.742	uns	uns	uns	.734	.886	.742	.967	uns	uns	uns

Table 1: Fraction of Discarded Work — Uniformly-Distributed Parameters

approach to schedule these types of tasks in general is to schedule only the extended mandatory parts of all but the last component task and then compensate for the accumulated error by giving the last component task as much time as possible. This approach is what Step 2 of all the DIST-* algorithms does, thus explaining the near-identical performance.

The only instance where the algorithms do not perform identically when h_i are small is when o_i and k_i are small as well. In this case, the approach described above takes about as much time as scheduling the component tasks precisely because all the optional parameters are small. In other words, the time given to the last component task to compensate for error in the mandatory parts (i.e., the mandatory extensions) and for the accumulated error of its predecessors is comparable to the amount of time it takes to simply execute all the shorter optional parts entirely to begin with. Because neither effect is pronounced, the heuristics in Steps 3 or 4 of the DIST-* algorithms can come into play and possibly produce better schedules. For instance, for small h_i, o_i, and k_i in Experiment 1, DIST-M$^+$ and DIST-M$^+$-ITERATIVE achieve the fraction of discarded work of 0.211 and perform better than the other algorithms.

In general, whenever h_i are small and o_i are large, the better approach is to schedule only the extended mandatory parts and then compensate for the accumulated error by giving the last component task as much time as available. On the other hand, whenever the o_i's or k_i's are small, the better approach is to schedule all the component tasks precisely. The performance of the five heuristics seen in all the experiments where the o_i's or k_i's are small supports this hypothesis. The heuristic steps in the DIST-* algorithms effectively "reconcile" these opposing approaches and determine how much of each scheduling approach to use in finding an assignment for each set of parameter values.

In summary, Algorithms DIST-M$^+$ and DIST-M$^+$-ITERATIVE perform better than the other algorithms in most cases; however in some cases, they do not. For example, in Experiment 4 when h_i, o_i, and k_i are small, DIST-M performs better than both DIST-M$^+$ and DIST-M$^+$-ITERATIVE because the locally optimal decisions made by DIST-M$^+$ and DIST-M$^+$-ITERATIVE were not globally optimal. Finally, the results of this set of experiments do not distinguish the performance of DIST-M$^+$ and DIST-M$^+$-ITERATIVE.

6.2 Bimodal Distribution

Table 2 shows the results obtained in four representative experiments where the parameters m_i, h_i, o_i, and k_i were chosen from either uniform or bimodal distributions. Specifically, in each experiment, the parameters of each component task were selected from either a uniform distribution in the range $[0, 100)$ or a bimodal distribution in the ranges $[0, 10)$ or $[90, 100)$. (The bimodal probability density function is equal to $1/20$ in the ranges $[0, 10)$ and $[90, 100)$ and is equal to zero elsewhere.) For example, when the h_i parameters are chosen from the bimodal distribution and m_i, k_i, and o_i are chosen from the uniform distribution, we label the appropriate column with "bimodal h."

The results in Table 2 show that the different heuristics used in the different DIST-* algorithms lead to a larger difference in performance when all or some of the task parameters are bimodally distributed. The reason is that Steps 1 and 2 do not often succeed for tasks with bimodally distributed parameters. For instance, while column 2 of Table 1 shows that when m_i, o_i, and k_i are large and h_i is small, all five DIST-* algorithms achieve the same result; the corresponding case of large m_i, o_i, and k_i chosen from the same uniform distribution but h_i chosen from the bimodal distribution (column 2 of Table 2) shows that the DIST-M$^+$ and DIST-M$^+$-ITERATIVE algorithms produce better results than the other algorithms. The bimodal mixture of small and large h_i makes the approaches taken by Step 1 and Step 2 less effective, thus allowing the heuristics of the DIST-M$^+$ and DIST-M$^+$-ITERATIVE algorithms to be applied more frequently and leading to better results. That Steps 1 and 2 fail more frequently for tasks with bimodally distributed parameters is demonstrated by the larger number of italicized entries in Table 2.

As in Table 1 when o_i or k_i are small, Table 2 shows that when o_i or k_i are bimodally distributed, the approach of scheduling all the component tasks precisely is still a good one. However, it is no longer as good as in the case when all task parameters are uniformly distributed; this is due mainly to the fact that fewer of the o_i and k_i parameters are small in the bimodal distribution when compared to the uniform distribution for a given composite task. Because the sample of o_i and k_i parameters is no longer uniformly distributed, the heuristics potentially have more opportunities to tradeoff the processing times of the mandatory and optional extensions of the component tasks, thus producing better results.

In summary, as in Table 1, Algorithms DIST-M$^+$ and DIST-M$^+$-ITERATIVE generally

Experiment 1

Algorithm	bimodal														
DIST-	*m,h,o,k*	*h*	*h,k*	*h,o*	*h,o,k*	*k*	*o*	*o,k*	*m*	*m,h*	*m,h,k*	*m,h,o*	*m,k*	*m,o*	*m,o,k*
M	*.000*	.920	*.932*	uns	*.925*	.948	uns	*.993*	.834	.950	*.358*	uns	*.663*	uns	*.961*
M^+	*.000*	*.834*	*.932*	*.946*	*.844*	.948	uns	*.930*	.834	*.865*	*.358*	*.569*	*.663*	uns	*.898*
M^+-ITER	*.000*	*.834*	*.932*	*.946*	*.844*	.948	uns	*.930*	.834	*.865*	*.358*	*.569*	*.663*	uns	*.898*
O	uns	.920	uns	uns	uns	.948	uns	uns	.834	.950	uns	uns	.872	uns	uns
O^+	uns	.920	uns	uns	uns	.948	uns	uns	.834	.950	uns	uns	.872	uns	uns

Experiment 2

Algorithm	bimodal														
DIST-	*m,h,o,k*	*h*	*h,k*	*h,o*	*h,o,k*	*k*	*o*	*o,k*	*m*	*m,h*	*m,h,k*	*m,h,o*	*m,k*	*m,o*	*m,o,k*
M	*.020*	uns	*.958*	*.789*	*.816*	*.797*	uns	*.946*	*.990*	uns	*.670*	*.714*	*.804*	uns	*.870*
M^+	*.058*	*.839*	*.958*	*.899*	*.854*	*.797*	uns	*.946*	*.931*	*.955*	*.670*	*.824*	*.804*	uns	*.870*
M^+-ITER	*.020*	*.839*	*.958*	*.789*	*.816*	*.797*	uns	*.946*	*.931*	*.955*	*.670*	*.714*	*.804*	uns	*.870*
O	uns	uns	uns	uns	uns	uns	uns	uns	uns	uns	uns	uns	uns	uns	uns
O^+	uns	uns	uns	uns	uns	uns	uns	uns	uns	uns	uns	uns	uns	uns	uns

Experiment 3

Algorithm	bimodal														
DIST-	*m,h,o,k*	*h*	*h,k*	*h,o*	*h,o,k*	*k*	*o*	*o,k*	*m*	*m,h*	*m,h,k*	*m,h,o*	*m,k*	*m,o*	*m,o,k*
M	*.154*	*.832*	*.154*	*.939*	*.062*	.944	uns	*.887*	.735	*.787*	*.808*	*.694*	.932	uns	*.642*
M^+	*.154*	*.832*	*.154*	uns	*.062*	.944	*.826*	*.887*	.735	*.787*	*.808*	*.914*	.932	*.791*	*.642*
M^+-ITER	*.154*	*.832*	*.154*	*.761*	*.062*	.944	*.826*	*.887*	.735	*.787*	*.808*	*.516*	.932	*.791*	*.642*
O	uns	.942	uns	uns	uns	.944	uns	uns	.735	.897	uns	uns	.932	uns	uns
O^+	uns	.942	uns	uns	uns	.944	uns	uns	.735	.897	uns	uns	.932	uns	uns

Experiment 4

Algorithm	bimodal														
DIST-	*m,h,o,k*	*h*	*h,k*	*h,o*	*h,o,k*	*k*	*o*	*o,k*	*m*	*m,h*	*m,h,k*	*m,h,o*	*m,k*	*m,o*	*m,o,k*
M	*.592*	*.559*	*.865*	uns	uns	.932	uns	*.849*	.859	*.836*	*.687*	uns	.873	uns	*.873*
M^+	*.538*	*.559*	*.865*	uns	*.947*	.932	*.814*	*.849*	.859	*.836*	*.687*	uns	.873	*.730*	*.873*
M^+-ITER	*.538*	*.559*	*.865*	*.721*	*.947*	.932	*.814*	*.849*	.859	*.836*	*.687*	*.637*	.873	*.730*	*.873*
O	uns	.808	uns	uns	uns	.932	uns	uns	.859	.909	uns	uns	.873	uns	uns
O^+	uns	.808	uns	uns	uns	.932	uns	uns	.859	.909	uns	uns	.873	uns	uns

Table 2: Fraction of Discarded Work — Bimodal & Uniformly-Distributed Parameters

perform better than the other algorithms when some parameters are bimodally distributed. Furthermore, DIST-M$^+$-ITERATIVE always performed as well as or better than DIST-M$^+$.

7 Summary and Future Work

In this paper, we extended the imprecise-computation technique to account for input error and end-to-end timing constraints and developed five heuristic scheduling algorithms — DIST-M, DIST-M$^+$, DIST-M$^+$-ITERATIVE, DIST-O, and DIST-O$^+$ — to minimize the output error of each composite task in a real-time system. The algorithms all have time complexity $O(n)$. Although our initial intuition indicated that the DIST-O and DIST-O$^+$ algorithms would perform better when the processing times of mandatory extensions were small compared to that of the optional extensions, this turned out to be false. For the suite of simulation experiments that were run to evaluate these algorithms, the DIST-M algorithm and its variants always performed as well as or better than either Algorithm DIST-O or Algorithm DIST-O$^+$. This makes sense because the optional extension does not have to be executed in any given schedule whereas the mandatory extension must always be executed. When an optional extension is "too long" (e.g., k_i and/or o_i large), the algorithms choose not to schedule it. Furthermore, when the mandatory extensions are small (i.e., h_i and/or m_i small), it is easier to compensate for the accumulated error produced by earlier component tasks by scheduling the last component task as much as possible.

As stated in Section 5, we could have used a linear-programming solver to do time distribution for the special case where the optional error-scaling factors are all zero. However, when the linear-programming solutions are applied to the composite tasks in the experimental suite (all of which have nonzero optional error-scaling factors), the solution provided by the linear-programming solver gives us an infeasible time distribution for every composite task generated in our experiments.

There are many natural extensions of this work including the generation of other precedence constraint graphs and more accurate piecewise linear approximations of the extension functions. For example, we are evaluating the performance of our algorithms when the extension functions of component tasks are concave and convex. Yet another extension would make more of a global decision in assigning time to the component tasks of a composite task. Specifically, when the low-level algorithm is unable to schedule the component tasks of a composite task within its allotted time, the low-level algorithm could provide feedback to the high-level algorithm about its additional processing-time needs. If the high-level algorithm finds other composite tasks that do not require the entirety of their assigned processing times, the unschedulable composite task could borrow time from them in order to become schedulable.

8 Acknowledgment

This work was supported in part by ONR contract no. N00014-92-J-1146 and AFOSR contract no. F49620-93-1-0060.

References

[1] A. L. Liestman and R. H. Campbell. A fault-tolerant scheduling problem. *IEEE Transactions on Software Engineering*, SE-12(10):1089–1095, October 1986.

[2] J. Blazewicz and G. Finke. Minimizing mean weighted execution time loss on identical and uniform processors. *Information Processing Letters*, 24(3):259–263, March 1987.

[3] K.-J. Lin and S. Natarajan. Concord: A system of imprecise computations. In *Proceedings of the 1987 IEEE Compsac*, pages 75–81, October 1987.

[4] K.-J. Lin, S. Natarajan, and J. W.-S. Liu. Imprecise results: Utilizing partial computations in real-time systems. In *Proceedings of 8th IEEE Real-Time Systems Symposium*, December 1987.

[5] J. W.-S. Liu, K.-J. Lin, and C. L. Liu. A position paper for the 1987 IEEE workshop on real-time operating systems. In *Proceedings of the 1987 IEEE Workshop on Real-Time Operating Systems*, May 1987.

[6] E. K. P. Chong and W. Zhao. Performance evaluation of scheduling algorithms for imprecise computer systems. Technical Report SA-5001, Department of Computer Science, University of Adelaide, September 1988.

[7] J. Y.-T. Leung, T. W. Tam, C. S. Wong, and G. H. Young. Minimizing mean flow time with error constraints. In *Proceedings of 10th IEEE Real-Time Systems Symposium*, pages 2–11, December 1989.

[8] J.-Y. Chung, J. W.-S. Liu, and K.-J. Lin. Scheduling periodic jobs that allow imprecise results. *IEEE Transactions on Computers*, 19(9):1156–1173, September 1990.

[9] J. Y.-T. Leung and C. S. Wong. Minimizing the number of late tasks with error constraints. In *Proceedings of 11th IEEE Real-Time Systems Symposium*, pages 32–40, December 1990.

[10] J. W.-S. Liu, K.-J. Lin, W.-K. Shih, A. C.-S. Yu, J.-Y. Chung, and W. Zhao. Algorithms for scheduling imprecise computations. *IEEE Computer*, 24(5):58–68, May 1991.

[11] W.-K. Shih, J. W.-S. Liu, and J.-Y. Chung. Algorithms for scheduling imprecise computations to minimize total error. *SIAM Journal on Computing*, 20(3), July 1991.

[12] W.-K. Shih and J. W.-S. Liu. On-line scheduling of imprecise computations to minimize error. In *Proceedings of the 13th IEEE Real-Time Systems Symposium*, December 1992.

[13] I. K. Cheong. Scheduling imprecise hard real-time jobs with cumulative error. Technical Report UIUCDCS-R-92-1758, Department of Computer Science, University of Illinois at Urbana-Champaign, June 1992.

[14] W. Zhao, J. W.-S. Liu, and S. Vrbsky. Imprecise scheduling in multiprocessor systems. In *Proceedings of the 5th IEEE International Conference on Parallel and Distributed Computing Systems*, September 1992.

[15] W.-K. Shih and J. W.-S. Liu. Algorithms for scheduling imprecise computations with timing constraints to minimize maximum error. *To appear in IEEE Transactions on Computers.*

[16] T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceedings of AAAI*, pages 49–54, 1988.

[17] M. Boddy and T. Dean. Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 67:245–285, 1994.

[18] S. Zilberstein. Constructing utility-driven real-time systems using anytime algorithms. In *Proceedings of the 1st IEEE Workshop on Imprecise and Approximate Computation*, December 1992.

[19] S. Zilberstein. *Operational Rationality Through Compilation of Anytime Algorithms*. PhD thesis, Department of Computer Science, University of California at Berkeley, 1993.

[20] S. Zilberstein. Optimizing decision qaulity with contract algorithms. In *Proceedings of IJCAI-95*, 1995.

[21] D. J. Musliner. *CIRCA: The Cooperative Intelligent Real-Time Control Architecture*. PhD thesis, Department of Computer Science and Engineering, University of Michigan, 1993.

[22] J. Dey, J. Kurose, and D. Towsley. On-line processor scheduling for a class of iris real-time tasks. *To appear in IEEE Transactions on Computers.*