Scheduling imprecise computation tasks on uniform processors

Guohua Wan\textsuperscript{a}, Joseph Y.-T. Leung\textsuperscript{b,∗}, Michael L. Pinedo\textsuperscript{c}

\textsuperscript{a} College of Management, Shenzhen University, Shenzhen 518060, China
\textsuperscript{b} Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102, USA
\textsuperscript{c} Stern School of Business, New York University, 40 West Fourth Street, New York, NY 10012, USA

Received 27 April 2006
Available online 18 May 2007
Communicated by F. Dehne

Abstract

We consider the problem of preemptively scheduling $n$ imprecise computation tasks on $m \geq 1$ uniform processors, with each task $T_i$ having two weights $w_i$ and $w'_i$. Three objectives are considered: (1) minimizing the maximum $w'$-weighted error; (2) minimizing the total $w$-weighted error subject to the constraint that the maximum $w'$-weighted error is minimized; (3) minimizing the maximum $w'$-weighted error subject to the constraint that the total $w$-weighted error is minimized. For these objectives, we give polynomial time algorithms with time complexity $O(mn^4)$, $O(mn^4)$ and $O(kmn^4)$, respectively, where $k$ is the number of distinct $w$-weights. We also present alternative algorithms for the three objectives, with time complexity $O(cm^3)$, $O(cm^3 + mn^4)$ and $O(kcmn^3)$, respectively, where $c$ is a parameter-dependent number.

© 2007 Elsevier B.V. All rights reserved.

Keywords: Preemptive scheduling; Uniform processors; Imprecise computation task; Polynomial time algorithms; Algorithms; Analysis of algorithms; Combinatorial problems

1. Introduction

The imprecise computation model was introduced to model applications in which the accuracy of computation can be sacrificed to meet deadline constraints of tasks in real time environments [9–12]. In the imprecise computation model, a task consists of two subtasks: a mandatory subtask and an optional subtask. Each task must have its mandatory part completed before its deadline, while its optional part may be left unfinished. If a part of the optional subtask remains unfinished, an error is incurred that is equal to the execution time of the unfinished part. The rationale of the imprecise computation model is to be able to trade-off computational accuracy versus the meeting of deadlines.

1.1. Problem description and notations

In the imprecise computational model, a task system $TS$ has $n$ tasks $T_1, T_2, \ldots, T_n$. A task $T_i$ is represented by a quadruple $T_i = \{r_i, d_i, m_i, o_i\}$, where $r_i, d_i, m_i,$ and $o_i$ denote the release time, deadline, mandatory execution time and optional execution time, respectively. Let $e_i = m_i + o_i$. A schedule is feasible if in this schedule, each task is executed between its release time and deadline, and it has received a processing time that is equal to at least its mandatory execution time and at most its
total execution time. If a feasible schedule exists, then task system \( TS \) is said to be schedulable. The feasibility of a task system can be checked by using the algorithm in [4]. Thus, we may assume that a given task system is feasible. Furthermore, we may assume that all the task parameters are rational numbers and that any task may be preempted by any other task during its execution.

To reflect the relative importance of the tasks, each task \( Ti \) is associated with two positive weights \( w_i \) and \( w_i' \). The task \( Ti \) can then be represented by a six-tuple \( Ti = (r_i, d_i, m_i, o_i, w_i, w_i') \).

A task system \( TS = (\{r_i\}, \{d_i\}, \{m_i\}, \{o_i\}, \{w_i\}, \{w_i'\}) \), is referred to as a doubly-weighted task system. Let \( S \) be a feasible schedule for \( TS \). We use the following notation and terminology.

\( p(T_i, S) \): the total amount of processing time assigned to task \( T_i \) in \( S \).

\( \epsilon(T_i, S) = e_i - p(T_i, S) \): the error of task \( T_i \) in schedule \( S \), i.e., the unexecuted portion of task \( T_i \) in \( S \).

\( \epsilon_w(T_i, S) = w_i \times \epsilon(T_i, S) \): the \( w \)-weighted error of task \( T_i \) in schedule \( S \).

\( \epsilon(S) = \sum_{i=1}^{n} \epsilon(T_i, S) \): the total error of \( TS \) in schedule \( S \).

\( \epsilon_w(S) = \sum_{i=1}^{n} \epsilon_w(T_i, S) \): the total \( w \)-weighted error of \( TS \) in schedule \( S \).

\( \epsilon_w(TS) = \min\{\epsilon_w(S) : S \) is a feasible schedule for \( TS \}\): the minimum total \( w \)-weighted error of \( TS \).

\( \epsilon_w(S) = \min\{\epsilon_w(T_i, S) : S \) is a feasible schedule for \( TS \}\): the maximum \( w \)-weighted error of \( TS \) in schedule \( S \).

\( \epsilon_w(TS) = \min\{\epsilon_w(S) : S \) is a feasible schedule for \( TS \}\): the minimum of the maximum \( w \)-weighted error of \( TS \).

\( \epsilon'_w(TS) : \) the minimum of the total \( w \)-weighted error subject to the constraint that the maximum \( w' \)-weighted error is minimized.

\( E_w'(TS) : \) the minimum of the maximum \( w' \)-weighted error subject to the constraint that the total \( w \)-weighted error is minimized.

1.2. Previous work

Blazewicz [1] introduced the problem of minimizing the total \( w \)-weighted error in case each task consists of only an optional subtask, i.e., \( m_i = 0 \) for all \( i = 1, 2, \ldots, n \). He solved the cases with a single processor, with identical processors in parallel, as well as with uniform processors in parallel by formulating the respective problems as linear programs. Blazewicz and Finke [2] solved the problem with identical processors in parallel as well as with uniform processors in parallel, by formulating the problem as min-cost-max-flow problems. They obtained an \( O(n^4 \log n) \)-time algorithm for identical processors and an \( O(m^2 n^4 \log mn + m^2 n^3 \log^2 mn) \)-time algorithm for uniform processors. Later, several researchers generalized these models and developed new algorithms for these problems. The results are summarized in Table 1. Note that in Table 1, [\(*\)] refers to this paper.

For the problem of minimizing the total unweighted error, Shih et al. [15] and Potts and Van Wassenhove [13] independently obtained \( O(n \log n) \)-time algorithms for the single processor environment. Shih et al. [15] also gave an \( O(n^2 \log n) \)-time algorithm for the total \( w \)-weighted error problem on a single processor. Later, Leung et al. [8] developed a more efficient algorithm, with time complexity \( O(n \log n + kn) \), where \( k \) is the number of \( w \)-distinct weights. Shih et al. [16] considered the multiprocessor case and gave algorithms for the unweighted and \( w \)-weighted cases with time complexity \( O(n^2 \log^2 n) \) and \( O(n^2 \log^3 n) \), respectively; see also [7]. Recently, Shakhlevich and Strusevich [14] considered the uniform processor case. They formulated the problem as the maximization of a linear function over...
a generalized polymatroid and developed an algorithm with time complexity $O(mn^4)$ for the total $w$-weighted error problem.

While minimizing the total $w$-weighted error is useful, the error may sometimes be unevenly distributed among the tasks, which is not desirable for a task system. Therefore, it is preferable to consider minimizing the maximum $w'$-weighted error of all the tasks. In this regard, Ho et al. [6] presented an $O(n^2)$-time algorithm for a single processor and an $O(n^2 \log^2 n)$-time algorithm for multiprocessors. Later, Choi et al. [3] improved their results for a single processor, presenting an algorithm with time complexity $O(n \log n + cn)$, where $c$ is a parameter-dependent number. Ho et al. [6] also considered the problem of minimizing the total $w$-weighted error, subject to the constraint that the maximum $w'$-weighted error is minimized. They developed algorithms with time complexity $O(n^3)$ and $O(n^3 \log^2 n)$ for a single processor and multiprocessors, respectively. Later, Ho and Leung [5] considered the problem of minimizing the maximum $w'$-weighted error, subject to the constraint that the total $w$-weighted error is minimized. They developed algorithms with time complexity $O(kn^2)$ and $O(kn^3 \log^2 n)$ for a single processor and multiprocessors, respectively, where $k$ is the number of distinct $w$-weights.

In this Letter, we shall generalize the results obtained so far in the literature to include the case of uniform processors in parallel. Specifically, we shall consider the preemptive scheduling of a doubly-weighted task system on uniform processors with objectives:

(i) the maximum $w'$-weighted error,
(ii) the total $w$-weighted error subject to the constraint that the maximum $w'$-weighted error is minimized, and
(iii) the maximum $w'$-weighted error subject to the constraint that the total $w$-weighted error is minimized.

2. The algorithms

In this section, we describe three algorithms for our scheduling problems with uniform processors. All three algorithms follow the same framework as those presented in [5,6]. Given a task system $TS = ([r_i], \{d_i\}, \{m_i\}, \{a_i\}, \{w_i\}, \{w'_i\})$, let

$$\min \{r_i\} = t_0 < t_1 < \cdots < t_p = \max \{d_i\}$$

be all the distinct values of the multiset \{r_1, \ldots, r_n, d_1, \ldots, d_n\}. These $p + 1$ values divide the time frame into $p$ intervals $[t_0, t_1], [t_1, t_2], \ldots, [t_{p-1}, t_p]$, denoted by $u_1, u_2, \ldots, u_p$. The length of the interval $u_j$, denoted by $l_j$, is defined as $t_j - t_{j-1}$. Furthermore, we introduce the following terminology; see [6].

A saturated interval: An interval $u_j$ is said to be saturated if at least one of the uniform processors has some idle time in $u_j$; otherwise, it is said to be unsaturated.

A fully scheduled task: A task $T_i$ is said to be fully scheduled in the interval $u_j$ if the entire interval $u_j$ is assigned to $T_i$; otherwise, it is said to be partially scheduled in $u_j$.

A precisely scheduled task: A task $T_i$ is said to be precisely scheduled in $S$ if $\varepsilon(T_i, S) = 0$; otherwise it is said to be imprecisely scheduled in $S$.

A removable task: A task $T_i$ is said to be removable if $a_i > 0$; otherwise it is said to be unremovable.

A shrinkable task: A task $T_i$ is said to be shrinkable if $a_i > 0$; otherwise it is said to be unshrinkable.

2.1. Maximum weighted error

First, we describe an algorithm, Algorithm A, for the problem of minimizing the maximum $w'$-weighted error of a doubly-weighted task system. Algorithm A proceeds in phases. In phase $l$, it constructs a task system $TS(l)$ from a task system $TS(l-1)$, the one from previous phase ($TS(0)$ is initialized to be $TS$). Then it apportions $\varepsilon(TS(l-1))$ amount of error to all the shrinkable tasks in $TS(l)$ in such a way that the maximum $w'$-weighted error is minimized. This is done by shrinking the optional subtask of task $T_i$ in $TS(l)$ by the amount $\varepsilon(TS(l-1))/\Delta$, where

$$\Delta = \sum_{i:a_i(l)>0} \frac{1}{w'_i},$$

where $a_i(l)$ denotes the shrinkable part of task $T_i$ in task system $TS(l)$ before the shrinking takes place in the $l$th phase. Task $T_i$ is said to be unshrinkable if $a_i(l)$ is 0. If $\varepsilon(TS(l)) = 0$, the algorithm stops. Otherwise, it removes some (but not necessary all) removable tasks in $TS(l)$ before proceeding to the next phase. This is done by finding an unsaturated interval $u_j$ (if any) in $S(l)$ in which a set of nonempty set of tasks is partially scheduled and removing these tasks from $TS(l)$. The algorithm is given below.

Algorithm A

Input: A doubly-weighted task system $TS = ([r_i], \{d_i\}, \{m_i\}, \{a_i\}, \{w_i\}, \{w'_i\})$ with $n$ tasks and $m \geq 1$ uniform processors.

Output: A schedule \( S \) for \( TS \) with \( E_w'(S) = E_w'(TS) \).

Method:
1. \( \eta(0) \leftarrow 0; \) \( TS(0) \leftarrow TS; l \leftarrow 0. \)
2. Using the algorithm in [4], construct a schedule \( S(0) \) with minimum total error for \( TS(0) \).
3. If \( \epsilon(S(0)) = 0 \), then output \( \eta(0) \) and stop.
4. \( l \leftarrow l + 1; TS(l) \leftarrow TS(l - 1); \) \( \Delta \leftarrow \sum_{j:o_j(l) > 0} \frac{1}{w_j^*}. \)
5. For each shrinkable task \( T_i \in TS(l) \) do:
   \[ \begin{align*}
   y_i & \leftarrow \epsilon(S(l - 1))/\Delta; \\
   o_i(l + 1) & \leftarrow \max\{0, o_i(l) - y_i\}; \\
   \end{align*} \]
   If \( o_i(l + 1) = 0 \) then mark \( T_i \) as unshrinkable.
6. \( \eta(l) \leftarrow \eta(l - 1) + \epsilon(S(l - 1))/\Delta. \)
7. Using the algorithm in [4], construct a schedule \( S(l) \) with minimum total error for \( TS(l) \).
8. If \( \epsilon(S(l)) = 0 \), then output \( \eta(l) \) and stop.
9. Find an unsaturated interval \( u_j \) in \( S(l) \) in which a nonempty set of tasks is partially scheduled and remove these tasks from \( TS(l) \).

To determine the computational complexity of Algorithm A, we state some properties of removable tasks. The proofs of these properties are similar to those presented in [6] and are omitted here.

**Property 1.** Let \( T_r \) be a removable task in \( TS \). Then \( T_r \) must be precisely scheduled in any schedule \( S \) such that \( \epsilon(S) = \epsilon(TS) \).

**Property 2.** Let \( T_r \) be a removable task in \( TS \). Then \( E_{w'}(TS) = E_{w'}(TS - \{T_r\}) \).

**Property 3.** Let \( S \) be a schedule for \( TS \) such that \( \epsilon(S) = \epsilon(TS) \). If \( T_r \) is partially scheduled in an unsaturated interval in \( S \), then \( T_r \) must be a removable task.

Based on the above properties, it is easy to prove the following lemma.

**Lemma 1.** If Algorithm A does not terminate in the \( l \)th phase (i.e., \( \epsilon(S(l)) > 0 \), \( l > 1 \), then either a task is marked as unshrinkable in step (5), or a task is removed in step (9) of the \( l \)th phase.

With the above lemma, we can prove the following theorem.

**Theorem 1.** The time complexity of Algorithm A is \( O(mn^4) \).

**Proof.** By Lemma 1, in each phase of Algorithm A, a task is either marked as unshrinkable or removed. Thus, in the worst case, the algorithm terminates after \( (2n - 1) \) phases.

It is clear that in each phase, every step, with the exception of steps (2), (7) and (9), takes at most \( O(n) \) time. Steps (2) and (7) can be done in \( O(n^2) \) time [4]. Step (9) can be done in at most \( O(n^2) \) time since there are at most \( O(n) \) intervals and for each interval there are at most \( n \) tasks eligible in the interval. Thus, the time complexity of Algorithm A is \( O(mn^4) \). \( \Box \)

To show the correctness of the algorithm, we now state the following lemmas. The proofs of these lemmas are similar to those in [6].

**Lemma 2.** For any task system \( TS \), \( E_{w'}(TS) \geq \epsilon(TS)/\Delta \), where \( \Delta = \sum_{i:o_i > 0}\frac{1}{w_i^*} \).

**Lemma 3.** For any task system \( TS \), let \( \overline{TS} \) be derived from \( TS \) by resetting the optional execution time of \( T_i \) to \( \max\{0, o_i - \frac{x}{w_i} \} \), where \( 0 \leq x \leq E_{w'}(TS) \). Then \( E_{w'}(\overline{TS}) + x \leq E_{w'}(TS) \).

Using Properties 2 and 3 and Lemmas 2 and 3, we can prove the following lemma (cf. [6]).

**Lemma 4.** Suppose that Algorithm A terminates in the \( \ell \)th phase, where \( \ell \geq 1 \). Then, \( \eta(\ell) + E_{w'}(TS^+(\ell)) \leq E_{w'}(TS) \) for each \( 0 \leq l \leq \ell \), where \( TS^+(\ell) \) denotes the task system immediately after the shrinking operation of the task system \( TS(l) \) (step (5) of Algorithm A).

With Lemma 4, we can now show the correctness of Algorithm A.

**Theorem 2.** Algorithm A correctly constructs a schedule for \( TS \) with the minimum of the maximum \( w' \)-weighted error.

**Proof.** Assume that Algorithm A terminates in the \( \ell \)th phase for task system \( TS \). By the nature of Algorithm A, we have \( \epsilon(TS^+(\ell)) = 0 \). Thus, \( E_{w'}(TS^+(\ell)) = 0 \). By Lemma 4, we know that \( \eta(\ell) \leq E_{w'}(TS) \). Since \( \epsilon(TS^+(\ell)) = 0 \), we can conclude that if the optional execution time of each task \( T_i \) were reset to \( \max\{0, o_i - \eta(\ell)/w_i^* \} \), there would be a feasible schedule \( S \) in which every task is precisely scheduled. Obviously, \( S \) is a feasible schedule for \( TS \) with \( E_{w'}(S) \leq \eta(\ell) \). Therefore, we have \( \eta(\ell) \geq E_{w'}(S) \geq E_{w'}(TS) \), and consequently, \( \eta(\ell) = E_{w'}(TS) \). \( \Box \)
2.2. Constrained total weighted error

In this subsection, we describe an algorithm, Algorithm B, for the problem of minimizing the total $w$-weighted error of a doubly-weighted task system, subject to the constraint that the maximum $w'$-weighted error is minimized. The basic idea of Algorithm B is as follows. Let $TS$ be a doubly-weighted task system and $S$ be a feasible schedule for $TS$. If $S$ were such a schedule that $E_w(S) = E_w(TS)$, then each task $T_i$ in $TS$ must be assigned an execution time at least $p(T_i, S) = m_i + \max\{0, \alpha_i - E_w(TS)/w'_i\}$. Now we modify $TS$ to ensure that task $T_i$ is assigned at least $p(T_i, S)$ amount of execution time by resetting its mandatory execution time to $\alpha_i - p(T_i, S)$, respectively. We then invoke the algorithm in [14] to construct a schedule for the modified task system with the minimum total $w$-weighted error. The algorithm is given below.

**Algorithm B**

**Input:** A doubly-weighted task system $TS = \{(r_i), \{d_i\}, \{m_i\}, \{\alpha_i\}, \{w_i\}, \{w'_i\}\}$ with $n$ tasks and $m \geq 1$ uniform processors.

**Output:** A schedule $S$ for $TS$ with $E_w(S) = E_w'(TS)$.

**Method:**

1. Call Algorithm A to compute $E_w'(TS)$.

2. Construct a new task system $TS'$ from $TS$ as follows. For $i = 1, 2, \ldots, n$ do

   a. $o_i(TS') = o_i$; $m_i(TS') = m_i$.
   b. if $E_w'(TS) < w'_i \times o_i$, then
   c. $m_i(TS') = \alpha_i - E_w'(TS)/w'_i$ and
   d. $o_i(TS') = E_w'(TS) / w'_i$.

   End for

3. Call the algorithm in [14] to construct a schedule $S$ for $TS'$.

Now we consider the computational complexity of Algorithm B.

**Theorem 3.** The time complexity of Algorithm B is $O(mn^4)$.

**Proof.** Step (1) of Algorithm B takes $O(mn^4)$ time. Step (2) takes $O(n)$ time and step (3) takes $O(mn^4)$ time. Thus, the overall running time of Algorithm B is $O(mn^4)$. □

Similar to the proof in [6], we also have the following theorem.

**Theorem 4.** Algorithm B correctly constructs a schedule $S$ for $TS$ with $E_w(S) = E_w'(TS)$.

2.3. Constrained maximum weighted error

In this subsection, we describe an algorithm, Algorithm C, for the problem of minimizing the maximum $w'$-weighted error of a doubly-weighted task system, subject to the constraint that the total $w$-weighted error is minimized. The algorithm goes through $k$ phases, where $k$ is the number of distinct $w$-weights in $TS$. In each phase, a new task system with identical $w$-weights is constructed. As it turns out (Lemma 5 stated below), when the $w$-weights are identical, then $E_w'(TS) = E_w'(TS)$. Thus, we can invoke Algorithm A at every phase to construct a schedule.

The ideas of Algorithm C are as follows. Given a doubly-weighted task system $TS = \{(r_i), \{d_i\}, \{m_i\}, \{\alpha_i\}, \{w_i\}, \{w'_i\}\}$ with $n$ tasks, let $\bar{w}_1 \geq \bar{w}_2 \geq \cdots \geq \bar{w}_k$ be the $k$ distinct $w$-weights in $TS$. A task whose weight is $\bar{w}_i$ is called a $l$-task. To obtain a schedule $S$ with $E_w(S) = E_w'(TS)$, Algorithm C goes through $k$ phases. In each phase $l$, it constructs a task system, $TS(l) = \{(r_i(l)), \{d_i(l)\}, \{m_i(l)\}, \{\alpha_i(l)\}, \{w_i(l)\}, \{w'_i(l)\}\}$ with $n$ tasks, let $\bar{w}_1 > \bar{w}_2 > \cdots > \bar{w}_k$ be the $k$ distinct $w$-weights in $TS$. A task whose $w$-weight is $\bar{w}_i$ is called a $l$-task. To obtain a schedule $S$ with $E_w'(TS) = E_w'(TS)$, Algorithm C goes through $k$ phases. In each phase $l$, it constructs a task system, $TS(l) = \{(r_i(l)), \{d_i(l)\}, \{m_i(l)\}, \{\alpha_i(l)\}, \{w_i(l)\}, \{w'_i(l)\}\}$ for $1 \leq l \leq k$ with $n$ tasks (Phase 0 will have the task system $TS(0) = TS$). For each task $T_i$ in $TS$, there is a task $T_i(l)$ in $TS(l)$ such that $r_i(l) = r_i, d_i(l) = d_i, w_i(l) = w_i, w'_i(l) = w'_i$. For each $1 \leq l \leq k$, if $T_i(l)$ is a $l'$-task, where $1 \leq l' \leq l$, then $m_i(l)$ is set to be the total processor time assigned to $T_i$ in the final schedule for $TS$ and $\alpha_i(l)$ is set to zero. Otherwise, they are set to $m_i$ and 0, respectively. Finally, $m_i(k), 1 \leq i \leq n$, gives the total processor time assigned to $T_i$ in the final schedule for $TS$. The algorithm is given below.

**Algorithm C**

**Input:** A doubly-weighted task system $TS = \{(r_i), \{d_i\}, \{m_i\}, \{\alpha_i\}, \{w_i\}, \{w'_i\}\}$ with $n$ tasks and $m \geq 1$ uniform processors.

**Output:** A schedule $S$ for $TS$ with $E_w'(S) = E_w'(TS)$.

**Method:**

1. Sort the tasks in nonincreasing order of the $w$-weights and let $\bar{w}_1 > \bar{w}_2 > \cdots > \bar{w}_k$ be the $k$ distinct $w$-weights in $TS$.
2. $\eta \leftarrow 0$; $\eta \leftarrow 0$; $TS(0) \leftarrow TS$ except that $\alpha_i(0) \leftarrow 0$ for all $i = 1, 2, \ldots, n$.
3. For $l = 1, 2, \ldots, k$ do:
   a. Sort the tasks in nonincreasing order of the $w$-weights.
   b. $\eta \leftarrow \eta + 1$.
   c. $TS(l) \leftarrow TS$.
   d. $\alpha_i(l) \leftarrow 0$ for all $i = 1, 2, \ldots, n$.
   e. The algorithm is given below.

   **Algorithm C**

   **Input:** A doubly-weighted task system $TS = \{(r_i), \{d_i\}, \{m_i\}, \{\alpha_i\}, \{w_i\}, \{w'_i\}\}$ with $n$ tasks and $m \geq 1$ uniform processors.

   **Output:** A schedule $S$ for $TS$ with $E_w'(S) = E_w'(TS)$.

   **Method:**

   1. Sort the tasks in nonincreasing order of the $w$-weights and let $\bar{w}_1 > \bar{w}_2 > \cdots > \bar{w}_k$ be the $k$ distinct $w$-weights in $TS$.
   2. $\eta \leftarrow 0$; $\eta \leftarrow 0$; $TS(0) \leftarrow TS$ except that $\alpha_i(0) \leftarrow 0$ for all $i = 1, 2, \ldots, n$.
   3. For $l = 1, 2, \ldots, k$ do:
      a. Sort the tasks in nonincreasing order of the $w$-weights.
      b. $\eta \leftarrow \eta + 1$.
      c. $TS(l) \leftarrow TS$.
      d. $\alpha_i(l) \leftarrow 0$ for all $i = 1, 2, \ldots, n$.
Lemma 5. Algorithm C invokes Algorithm A $k$ times.
Proof. The correctness of the algorithm is based on the constraint that the total $w'$-weighted error is minimized. Its time complexity is $O(kmn^4)$, where $k$ is the number of distinct $w$-weights.

Based on the above property, we have the following theorem.

Theorem 5. Algorithm C constructs a schedule that minimizes the maximum $w'$-weighted error, subject to the constraint that the total $w$-weighted error is minimized. Its time complexity is $O(kmn^4)$, where $k$ is the number of distinct $w$-weights.

Proof. The correctness of the algorithm is based on Lemma 5. Algorithm C invokes Algorithm A $k$ times. Thus, its time complexity follows from the time complexity of Algorithm A, which is $O(mn^4)$.

3. Alternative algorithms

In this section, we describe an alternative algorithm for Algorithm A. To develop a theoretical basis for the algorithm, we first present two lemmas. Given a positive constant $\alpha > 0$, let $TS_\alpha$ be a new task system derived from $TS$ by setting the optional and mandatory execution time as follows:

1. $o_i(TS_\alpha) = 0$ for all tasks.
2. $m_i(TS_\alpha) = e_i - \frac{\alpha}{w_i'}$, if $\alpha < w_i' \times o_i$, and
3. $m_i(TS_\alpha) = m_i$, otherwise.

Now we describe two important properties of the scheduling problem (cf. [3]).

Lemma 6.
1. If $TS_\alpha$ is schedulable, then for any constant $\beta \geq \alpha$, $TS_\beta$ is also schedulable.
2. If $TS_\alpha$ is not schedulable, then for any constant $\beta \leq \alpha$, $TS_\beta$ is also not schedulable.

Proof. Since $\beta \geq \alpha$, the computational requirements of all tasks in $TS_\beta$ are less than or equal to those of the tasks in $TS_\alpha$. Hence if a task system $TS_\alpha$ is schedulable, $TS_\beta$ is also schedulable. Thus, (1) holds and (2) follows immediately from (1).

Lemma 7. For any positive constant $\alpha$, if $\alpha \geq E_w'(TS)$, where $E_w'(TS)$ is the minimum of the maximum $w'$-weighted error of $TS$, then $TS_\alpha$ is schedulable. In other words, $E_w'(TS)$ is the minimum of $\alpha$ such that $TS_\alpha$ is schedulable, i.e., if $\alpha < E_w'(TS)$, then $TS_\alpha$ is not schedulable.

Proof. Since $E_w'(TS)$ is the minimum of the maximum $w'$-weighted error, there exists a schedule $S$ such that $w'_i \times (e_i - p(T_i, S)) \leq E_w'(TS)$, $i = 1, \ldots, n$.
That is, $e_i - \frac{E_w'(TS)}{w'_i} \leq p(T_i, S)$, $i = 1, \ldots, n$.
Thus, $m_i(TS_{E_w'(TS)}) \leq p(T_i, S)$, $i = 1, \ldots, n$.
Hence, $S$ is still a feasible schedule for the task system $TS_{E_w'(TS)}$, i.e., $TS_{E_w'(TS)}$ is schedulable. According to Lemma 6, $TS_\alpha$ is schedulable for any $\alpha > E_w'(TS)$, and, for any $\alpha < E_w'(TS)$, $TS_\alpha$ is not schedulable. Therefore, $E_w'(TS)$ is the minimum of $\alpha$ such that $TS_\alpha$ is schedulable.

From the above two properties, we know that the minimum of the maximum $w'$-weighted error of a task system is equal to the smallest positive constant (let it be $\alpha$) such that the derived task system $T_\alpha$ is not schedulable below this number.

The ideas underlying the proposed algorithm can be sketched as follows. The minimum of the maximum $w'$-weighted error for a task system is determined by conducting a binary search on the minimum value of $\alpha$ such that $TS_\alpha$ is schedulable. The range of the binary search will be $[0, \max_{1 \leq i \leq n} \{w'_i \times o_i\}]$. For every value $\alpha$ obtained in the binary search, we call the algorithm in [4] to determine if $TS_\alpha$ is schedulable. Since the exact value of $E_w'(TS)$ (which is the minimum value of $\alpha$) may
Algorithm D

Input: A doubly-weighted task system $TS = ([r_i], [d_i], [m_i], [o_i], [w_i], [w_i'])$ with $n$ tasks and $m \geq 1$ uniform processors.

Output: A schedule $S$ for $TS$ such that the maximum $w'$-weighted error of $S$ is within the ErrorBound of $E_{w'}(TS)$.

Method:

1. $low \leftarrow 0$ and $high \leftarrow \max_{1 \leq i \leq n} \{w_i' \times o_i\}$.
   1. $\alpha \leftarrow \frac{(high - low)}{2}$.
   2. Construct a task system $TS_\alpha$ as follows:
      - For $i = 1, 2, \ldots, n$ do
        - $o_i(TS_\alpha) \leftarrow 0$;
        - if $\alpha < w_i' \times o_i$, then
          - $m_i(TS_\alpha) \leftarrow e_i - \frac{\alpha}{w_i'}$;
        - else $m_i(TS_\alpha) = m_i$.
      End for
   3. Check whether $TS_\alpha$ is schedulable or not by using the algorithm in [4], and then modify the values of low and high as follows:
      - if $TS_\alpha$ is schedulable, then $high \leftarrow \alpha$;
      - else $low \leftarrow \alpha$.

2. Step 2 may iterate at most $c$ times, where
   $$c = \min \left\{ q: \text{ErrorBound} \leq \max_{1 \leq i \leq n} \{w_i' \times o_i\}/2^q \right\}.$$

Thus, it can be done in $O(cn^3)$ time. Step 3 is just one invocation of the algorithm in [4], which has time complexity $O(mn^3)$. Hence, the time complexity of the algorithm is $O(cn^3)$. Therefore, we have the following theorem.

Theorem 7. The time complexity of Algorithm D is $O(cn^3)$.

Remark. Note that alternative algorithms for Algorithms B and C can also be developed by invoking Algorithm D, instead of Algorithm A, in Algorithms B and C, respectively. Then the time complexity will be $O(cn^3 + mn^3)$ and $O(kcn^3)$, respectively, where $k$ and $c$ are defined as before.

4. Concluding remarks

In this paper, we have extended previous results of preemptively scheduling imprecise computation tasks on a single processor and parallel and identical processors to $m \geq 1$ uniform processors, for the three objectives: (1) minimizing the maximum $w'$-weighted error; (2) minimizing the total $w$-weighted error subject to the constraint that the maximum $w'$-weighted error is minimized; (3) minimizing the maximum $w''$-weighted error subject to the constraint that the total $w$-weighted error is minimized. For these objectives, we presented polynomial time algorithms, with time complexity $O(mn^4)$, $O(mn^4)$ and $O(kmn^4)$, respectively, where $k$ is the number of distinct $w$-weights. We also presented alternative algorithms for the three algorithms, with time complexity $O(cn^3)$, $O(cn^3 + mn^3)$ and $O(kcn^3)$, respectively, where $c$ is a parameter-dependent number.

For future research, it will be interesting to see if there are faster algorithms to solve any of the above scheduling problems.

Acknowledgements

The work of the first author is supported in part by the NSF of China (70372058) and NSF of Guangdong (031808). The work of the second author is supported in part by the NSF Grant DMI-0300156. The work of the third author is supported in part by the NSF Grant DMI-0245603.

References


