Heuristic Task Assignment for Distributed Computing Systems

CHIUN-CHIEH HSU

and

SHENG-DE WANG

*Department of Electrical Engineering, National Taiwan University,
Taipei, Taiwan, 10764, R.O.C.*

ABSTRACT

This paper addresses the problem of assigning a task with precedence constraint to a distributed computing system. The task turnaround time including communication overhead and idle time is adopted to measure the performance of task assignment. The task assignment in this paper requires one to determine not only the assignment of modules, but also the sequence of message transmission to balance processor loads and diminish communication overhead. The search for the optimal task assignment with precedence constraint is known to be NP-complete in the strong sense. A heuristic algorithm with polynomial time complexity is then proposed in order to effectively solve the task assignment problem. The experimental results reveal that the proposed approach is able to obtain a near-optimal or even the optimal task assignment.

I. INTRODUCTION

The rapid progress of VLSI and computer networking technologies has made distributed computing systems economically attractive for many computer applications. Although distributed computing systems are capable of enhancing system throughput and resource utilization, they raise some problems that prevent their widespread use. One of the major problems is the throughput degradation caused by imbalance of processor loads and the large amount of interprocessor communication overhead.

Therefore, the aim of task assignment in this paper is to attain the minimum response (task turnaround) time to maximize system throughput. There are two conflicting strategies to diminish the task turnaround time, namely, assigning all modules to a single processor to save communication overhead, and distributing them evenly to all processors to balance the load. A

compromise between the two strategies should be made to achieve the minimum task turnaround time. However, obtaining the minimum turnaround time for a partitioned task with precedence constraint is extremely difficult and generally intractable. If we cannot achieve the minimum task turnaround time within reasonable time, we hope to make use of some heuristics to attain a near-minimum one with much less effort.

Several approaches to the task assignment in distributed computing systems have been identified. They can be roughly classified into four categories: graph-theoretical [1–3], mathematical-programming [4–6], heuristic [7–12, 17, 19], and simulated-annealing [13]. In general, these approaches are not mutually exclusive. The graph-theoretical method models the problem as a processor-module flow graph and partitions it using a max-flow min-cut algorithm. The mathematical-programming approach formulates the problem as a combinatorial optimization problem and solves it using a mathematical optimization technique. The heuristic method utilizes heuristic information to evaluate the assignment cost and speeds up the procedure of finding an assignment. The simulated-annealing approach uses a stochastic search criterion to iteratively improve an initial assignment to a globally optimal task assignment.

Many of these researches ignore the precedence constraint, despite the fact that it is an important characteristic of a real programming environment. Even if the precedence constraint is included, communication overhead and idle time due to control overhead are generally neglected [10]. In this article, we present an approach which takes the precedence constraint, communication overhead, and idle time into account. In addition, the distributed computing system with non-fully-connected topology and multihop communication are also allowed.

In this paper, the task turnaround time is adopted to evaluate the effectiveness of assignment and is computed by using the concepts of trigger moment, activation moment, and start moment. The task assignment requires determining not only the assignment of modules, but also the sequence of message transmission. For the former it assigns modules to the best-fit processors, which leads to load balance; for the latter it determines the best sequences of parent modules transmitting messages to their children, which leads to diminishing communication overhead. Since pursuing the minimum task turnaround time (optimal task assignment) is known to be NP-complete in the strong sense [10, 15, 16], six heuristics are proposed to effectively assign modules and determine the transmission sequence. Although it is a heuristic method, experimental results show that a near-optimal or even the optimal task assignment will be achieved.

The remainder of this paper is organized as follows: Section II presents the problem statement. Trigger moment, activation moment, start moment, and the computation of task turnaround time are also defined in this section. Six heuristics are proposed in Section III. An illustrative example is given and

experimental results are discussed in Section IV. Finally, conclusions are drawn in Section V.

II. PROBLEM STATEMENT AND DEFINITION OF THE COST FUNCTION

A. PROBLEM STATEMENT

In this paper, a task is assumed to be partitioned into modules with precedence constraints. If module α immediately precedes module β , we say that α is the parent of β or β is the child of α . In the partitioned task, there is at least one module without parent and at least one module without child. These are called source and sink modules respectively. Directed cycles in a task are required to be broken or merged into larger modules before treatment according to this paper.

In this article, a distributed computing system is composed of homogeneous processors, and its topology is not necessarily fully connected. The approach proposed can still be applied in a heterogeneous system if average execution times and communication times are used in the proposed heuristics. The interprocessor link of a distributed computing system is bidirectional, and the communication times for transmitting a message from either end of a link to the other are the same. Compared to the communication time in an interprocessor link, the communication time for transmitting a message inside a processor is negligible. Therefore, if a parent and its child are assigned to the same processor, their communication time is assumed to be zero.

There exist two factors contributing to the cost of task assignment, namely, the assignment of modules and the sequences of parents transmitting messages to their children. They are called module assignment and the transmission sequence respectively. The former is very clear, but the latter requires further explanation. The interprocessor communication in this paper is not based on a common bus structure. Hence, the transmission sequence plays an important role in the process of a parent sending the required messages to its children to wake them up.

B. COMPUTATION OF THE COST FUNCTION

For module α and its child β , the *trigger moment* of α with respect to β is the instant after α has transmitted a message to β . A module β is activated if all of its parents have triggered it. Thus, the *activation moment* of β is the instant when the last parent of β finishes transmitting the required message to β . If β is activated and its assigned processor is released from any previous execution and communication, the processor starts executing β at that mo-

ment, which is called the *start execution moment* (*start moment* for short) of β . All terms described above imply the phenomenon of time accumulation as a consequence of precedence constraints between modules.

Let us introduce some more terms, which will be used in the remainder of this paper.

(1) The task assignment MT is composed of a module assignment M and a transmission sequence T .

(2) $M(\alpha)$ denotes the processor to which module α is assigned by module assignment M .

(3) $T_\alpha(i)$, the i th triggered child of α , is the i th child to which its parent α transmits a message.

(4) $RL_\alpha(MT)$ denotes the moment when $M(\alpha)$ is released from any previous execution and communication, and is ready to execute α .

(5) E_α denotes the elapsed time required for module α to be fully executed.

(6) $C_{\alpha,\beta}$ denotes the communication time interval required to send a message from α to β , which are assigned to adjacent processors. It is infinite if α is not the parent of β .

(7) $TIME_p$ denotes the current elapsed time recorded in processor p at any point during the computation of the cost function (task turnaround time). $TIME_p$ includes all execution times, communication times, and idle times.

Let us discuss how to compute the turnaround time for a totally assigned task. Initially, the trigger moments, activation moments, and start moments of all source modules are set to zero; likewise the elapsed times of all processors. Based on this initialization, the start moment of an assigned module and then the trigger moment of this module with respect to its children are computed. This procedure is repeated from source modules to sink modules.

Suppose there is a parent α which forks to n children. Assuming asynchronous IO (input-output), the trigger moments of α with respect to its children associated with a task assignment MT are computed with the following algorithm.

ALGORITHM 1.

/* The argument MT for the task assignment is dropped in this algorithm. SEND denotes the instant when a processor begins sending a message; ARRIVAL denotes the moment when the message arrives. */

- (1) $TIME_{M(\alpha)} = ST_\alpha + E_\alpha$, where ST_α denotes the start moment of α ;
- (2) FOR $i = 1$ TO n DO
- (3) BEGIN
- (4) IF (processors $M(\alpha)$ and $M(T_\alpha(i))$ are the same)

- (5) THEN $\text{TRG}_{\alpha, T_{\alpha}(i)} = \text{TIME}_{M(\alpha)}$;
- (6) ELSE IF $(M(\alpha)$ and $M(T_{\alpha}(i))$ are adjacent)
- (7) THEN $\text{TRG}_{\alpha, T_{\alpha}(i)} = \text{TIME}_{M(\alpha)} + C_{\alpha, T_{\alpha}(i)}$;
- (8) ELSE BEGIN
/* If the required message of α with respect to its i th triggered child $T_{\alpha}(i)$ is transmitted along processors $P_1, P_2, \dots, P_s$, where $M(\alpha) = P_1$, $M(T_{\alpha}(i)) = P_s$, and of course $P_j \neq P_k$ for $j \neq k$. */
(9) SEND = TIME_{P_1} ;
- (10) FOR $j = 1$ TO $s - 2$ DO
- (11) BEGIN
- (12) $\text{TIME}_{P_j} = \text{ARRIVAL} = \text{SEND} + C_{\alpha, T_{\alpha}(i)}$;
- (13) $\text{TIME}_{P_{(j+1)}} = \text{SEND} = \max\{\text{ARRIVAL}, \text{TIME}_{P_{(j+1)}}\}$;
- (14) ENDFOR;
- (15) $\text{TIME}_{(s-1)} = \text{TRG}_{\alpha, T_{\alpha}(i)} = \text{SEND} + C_{\alpha, T_{\alpha}(i)}$;
- (16) ENDIF;
- (17) ENDFOR;
- (18) END of Algorithm 1.

If these n children are assigned to the same processor as $M(\alpha)$ or the adjacent processors to $M(\alpha)$, then the trigger moment of α with respect to $T_{\alpha}(i)$ will be simplified to

$$\text{TRG}_{\alpha, T_{\alpha}(i)}(MT) = \text{ST}_{\alpha}(MT) + E_{\alpha} + \sum_{\substack{j=1 \\ M(\alpha) \neq M(T_{\alpha}(j))}}^i C_{\alpha, T_{\alpha}(j)}. \quad (1)$$

The last term in Equation (1) indicates that if the first i triggered children of α are assigned to the processors adjacent to α 's, their communication times will be added to the trigger moment of α with respect to $T_{\alpha}(i)$. While trigger moments are being computed, $\text{TIME}_{M(\alpha)}$ requires to be updated.

For module β and the set of its parents F_{β} , the activation moment and start moment of β are given by

$$\text{ACT}_{\beta}(MT) = \max_{\alpha \in F_{\beta}} \text{TRG}_{\alpha, \beta}(MT), \quad (2)$$

$$\text{ST}_{\beta}(MT) = \max\{\text{RL}_{\beta}(MT), \text{ACT}_{\beta}(MT)\}. \quad (3)$$

The processor elapsed time $\text{TIME}_{M(\beta)}$ also requires to be updated while the start moment of β is being computed.

Recall that an activated module β cannot be executed immediately if its assigned processor $M(\beta)$ has not yet been released. As a consequence, β will

Fig. 1.

cause a delay between the activation moment and release moment. This period of time is called the delay time of β , and is defined as

$$\text{DELAY}_{\beta}(MT) = \text{RL}_{\beta}(MT) - \text{ACT}_{\beta}(MT). \quad (4)$$

On the contrary, if β 's release moment is earlier than its activation moment, the processor $M(\beta)$ will be idle. The corresponding period of time is called the idle time of $M(\beta)$, and is defined as

$$\text{IDLE}_{M(\beta)} = \text{ACT}_{\beta}(MT) - \text{RL}_{\beta}(MT). \quad (5)$$

Let us use a simple example to show how to compute these terms. A distributed computing system and a task are modeled as an undirected graph and a directed acyclic graph as shown in Figures 1 and 2, respectively. There exist a module assignment $M = \{(A, 1), (B, 2), (C, 1), (D, 3)\}$ and a transmission sequence $T_A = \{B, C\}$, $T_B = \{D\}$, $T_C = \{D\}$, and $T_D = \{ \}$. All execution times are 30 time units (e.g. milliseconds), and all communication times are 5 time units. The steps required to compute these terms are as follows:

1. Initially, $\text{TIME}_1 = \text{TIME}_2 = \text{TIME}_3 = 0$, $\text{ACT}_A = \text{ST}_A = 0$.
2. First of all, source module A is executed in processor 1. Thus,

$$\text{TIME}_1 = \text{ST}_A + E_A = 30.$$

Fig. 2.

3. Then, A transmits messages to B first and then to C :

$$\text{TIME}_1 = \text{TRG}_{A,B} = 30 + 5 = 35,$$

$$\text{TIME}_1 = \text{TRG}_{A,C} = 35 + 0 = 35.$$

4. Two modules B and C are activated concurrently:

$$\text{ACT}_B = \text{ACT}_C = 35.$$

Module B transmits a message to D from processor 2 to 3:

$$\text{TIME}_2 = \text{ST}_B = \max\{\text{RL}_B, \text{ACT}_B\} = \max\{\text{TIME}_2, \text{ACT}_B\} = 35,$$

$$\text{TIME}_2 = \text{ST}_B + E_B = 65,$$

$$\text{TIME}_2 = \text{TRG}_{B,D} = 65 + 5 = 70.$$

Module C transmits a message to D from processor 1 through 2 to 3:

$$\text{TIME}_1 = \text{ST}_C = \max\{\text{RL}_C, \text{ACT}_C\} = \max\{\text{TIME}_1, \text{ACT}_C\} = 35,$$

$$\text{SEND} = \text{TIME}_1 = \text{ST}_C + E_C = 65,$$

$$\text{TIME}_1 = \text{ARRIVAL} = \text{SEND} = C_{C,D} = 65 + 5 = 70,$$

$$\text{TIME}_2 = \text{SEND} = \max\{\text{ARRIVAL}, \text{TIME}_2\} = 70,$$

$$\text{TRG}_{C,D} = \text{TIME}_2 = \text{SEND} + C_{C,D} = 75.$$

5. Finally,

$$\text{ACT}_D = \max\{\text{TRG}_{B,D}, \text{TRG}_{C,D}\} = 75,$$

$$\text{TIME}_3 = \text{ST}_D = \max\{\text{RL}_D, \text{ACT}_D\} = \max\{\text{TIME}_3, \text{ACT}_D\} = 75;$$

$$\text{TIME}_3 = \text{ST}_D + E_D = 105.$$

Note that MT has been dropped for clarity.

After all terms of the modules for a totally assigned task have been computed, the task turnaround time associated with a task assignment MT is then defined as

$$\text{TA}(MT) = \max_{p \in P} \text{TIME}_p(MT), \quad (6)$$

where P denotes the set of processors.

III. HEURISTIC TASK ASSIGNMENT

It is known that the search for an optimal task assignment with precedence constraint is NP-complete in the strong sense [10, 15, 16]. Heuristic methods have been considered as the most powerful approaches to circumvent this difficulty. Therefore, in this section, we will propose six heuristics combining a scheme of reassignment to minimize interprocessor communication overhead and balance processor loads, so we can achieve a suboptimal assignment with much less effort. Since this method is heuristic, deviation from the optimal solution is inevitable. However, experimental results reveal that the deviation is gratifyingly tolerable and even the optimal solution is often attained.

From Algorithm 1 and Equations (1) to (6) above, we know that transmission sequences affect the computation of trigger moments, activation moments, start moments, and the task turnaround time. Thus, we intend to assign modules to the best-fit processor as well as determine the best transmission sequence in order to obtain the minimum or a near-minimum task turnaround time.

The heuristic assignment presented in this section is divided into four parts:

- (1) initial transmission-sequence decision;
- (2) coassignment and final transmission-sequence decisions;
- (3) module assignment;
- (4) reassignment.

Part (1) concerns whether execution time or communication time will be adopted as the metric of transmission-sequence decision. Furthermore, an initial sequence will be chosen in some order of execution times or communication times. Part (2) proposes a method to delicately determine whether or not children are assigned to the same processor as their parents' (the coassignment decision), and thus decide the final transmission sequence. Hence, communication overhead can be minimized and processor loads can be balanced. In addition, if several modules are activated concurrently, their levels and estimated finish moments are used as a basis on which to construct the priority list in order to make coassignment and transmission-sequence decisions. Part (3) presents a heuristic to assign an unassigned module to the processor which leads to minimizing the task turnaround time. A simple but effective reassignment is suggested in part (4) to handle an abnormal condition resulting from bad task partitionings.

Before describing the heuristic task assignment, we define two terms as follows. Given several modules, the processors adjacent to all of their assigned processors are called their *one-hop processors*. If these modules are assigned to the same processor, we say that this processor is their *zero-hop processor*.

A. INITIAL TRANSMISSION-SEQUENCE DECISION

LEMMA 1. *If a parent forks to n children, there exist $n!$ different sequences for the parent to transmit messages to its children.*

LEMMA 2. *If a parent α forks to n children which are assigned to the zero-hop or one-hop processors of α , the total communication overhead associated with a task assignment MT is*

$$\sum_{j=1}^n (n-j+1)C_{\alpha, T_{\alpha}(j)}.$$

$M(\alpha) \neq M(T_{\alpha}(j))$

Proof. Equation (1) states that the trigger moment of α with respect to its i th triggered child $T_{\alpha}(i)$ is

$$\text{TRG}_{\alpha, T_{\alpha}(i)}(MT) = \text{ST}_{\alpha}(MT) + E_{\alpha} + \sum_{j=1}^i C_{\alpha, T_{\alpha}(j)}.$$

$M(\alpha) \neq M(T_{\alpha}(j))$

Thus, the total communication overhead is

$$\begin{aligned} & \sum_{i=1}^n \left\{ \text{TRG}_{\alpha, T_{\alpha}(i)}(MT) - \text{ST}_{\alpha}(MT) - E_{\alpha} \right\} \\ &= \sum_{i=1}^n \sum_{j=1}^i C_{\alpha, T_{\alpha}(j)} \\ & \quad M(\alpha) \neq M(T_{\alpha}(j)) \\ &= \sum_{j=1}^n \sum_{i=j}^n C_{\alpha, T_{\alpha}(j)} = \sum_{j=1}^n (n-j+1)C_{\alpha, T_{\alpha}(j)}. \quad \blacksquare \\ & \quad M(\alpha) \neq M(T_{\alpha}(j)) \end{aligned}$$

THEOREM 1. *Let there be a parent α forking to n children which are assigned to the one-hop processors of α . If these children are triggered in the nondecreasing order of communication time, the minimum total communication overhead will be attained among all $n!$ transmission sequences.*

Proof. Children are assigned to the one-hop processors of α , which implies $M(\alpha) \neq M(T_{\alpha}(j))$ for $1 \leq j \leq n$. Thus the term $M(\alpha) \neq M(T_{\alpha}(j))$ in Lemma 2 can be dropped. The total communication overhead turns out to be $\sum_{j=1}^n (n-j+1)C_{\alpha, T_{\alpha}(j)}$.

It is trivial to prove if $C_{\alpha, T_{\alpha}(j)} \leq C_{\alpha, T_{\alpha}(j+1)}$ for $1 \leq j < n$ (i.e., children are triggered in the nondecreasing order of communication time), we will obtain the minimum total communication overhead. ■

There are two factors contributing to transmission-sequence decision, namely, the amounts of execution time and communication time. We will propose a heuristic to determine whether execution time or communication time will be adopted to make the transmission-sequence decision.

HEURISTIC 1. Suppose a parent has been assigned and its children have not. If the sum of the children's execution times is larger than the minimum total communication overhead, execution time will be adopted as the metric for deciding the transmission sequence; otherwise, communication time will be adopted.

If execution time is used to make the decision of transmission sequence, then a child with larger execution time will be triggered earlier. From Equation (1), we know that the earlier a child is triggered, the smaller the trigger moment it has. As a consequence, summing the smaller trigger moment and the larger execution time will lead to even finish moments $[st_{\alpha}(MT) + E_{\alpha}]$ for children. In other words, the load of children's processors tends to be balanced. Hence, we have Heuristic 2 to rationally determine the initial transmission sequence.

HEURISTIC 2. If the metric for deciding the transmission sequence is execution time, children are initially triggered in the nonincreasing order of execution time.

If communication time is adopted, according to Theorem 1, a child with smaller communication time will be triggered earlier to achieve the minimum total communication overhead. Therefore, we have Heuristic 3.

HEURISTIC 3. If the metric for deciding the transmission sequence is communication time, children are initially triggered in the nondecreasing order of communication time.

In practice, children are first sorted according to Heuristic 3 to facilitate the computation of the minimum total communication overhead in Heuristic 1. Then, if execution time is adopted as the decision metric by using Heuristic 1, children will be re-sorted according to Heuristic 2; otherwise, children need not be re-sorted. The transmission sequence determined by Heuristic 2 or 3 is only an initial transmission sequence. Some subtle modifications will be made in the following heuristic.

B. COASSIGNMENT AND FINAL TRANSMISSION-SEQUENCE DECISION

Recall that if a child is assigned to the same processor as its parent, the communication time for message transmission can be saved. However, under this condition, the child will be executed on this commonly assigned processor, i.e., the child's execution time will be included in the elapsed time of the processor. The inclusion of the execution time may result in load imbalance of processors and offset the benefit of saving communication overhead. We now consider the problem whether or not to assign a child to the same processor as its parent. We will refer the child assigned to the same processor as its parent as a *coassigned* child. A coassigned child must wait for its parent's processor to send messages to other children and then process them, so that it will be triggered last.

Before describing the following theorem, we define the n/k fork as follows.

DEFINITION 1. Let there be a parent which forks to n children. We define it as an n/k fork if the last k triggered children are coassigned with the parent and others are assigned to the one-hop processors of the parent.

THEOREM 2. Let there be an n/k fork where the parent is α . If the i th triggered child, $1 \leq i \leq n - k$, is now coassigned and triggered last, the total amount of variation of trigger moments for α with respect to its children is

$$\sum_{j=i+1}^{n-k} C_{\alpha, T_{\alpha}(j)} - (n - i + 1)C_{\alpha, T_{\alpha}(i)}.$$

Proof. Let $M'T'$ denote the modified task assignment. The relationship between T and T' is $T_{\alpha}(j) = T'_{\alpha}(j)$ for $1 \leq j < i$, $T_{\alpha}(i) = T'_{\alpha}(n)$ and $T_{\alpha}(j) = T'_{\alpha}(j - 1)$ for $i + 1 \leq j \leq n$. Furthermore, $M'(T_{\alpha}(j)) = M(T_{\alpha}(j))$ for $1 \leq j \leq n$ and $j \neq i$.

The proof is divided into three parts as follows:

(1) Since children $T_{\alpha}(j)$ for $1 \leq j < i$ are triggered before $T_{\alpha}(i)$, the retrigger and coassignment of $T_{\alpha}(i)$ do not affect its trigger moment. That is, $\text{TRG}_{\alpha, T'_{\alpha}(j)}(M'T') = \text{TRG}_{\alpha, T_{\alpha}(j)}(MT)$. Thus, the amount of variation of trigger moments for the first $i - 1$ triggered children is zero.

(2) Since the first $n - k$ triggered children other than $T_{\alpha}(i)$ are assigned to the one-hop (adjacent) processors of α and the last k triggered children are coassigned with α , the trigger moment of α with respect to $T_{\alpha}(i)$ after

retrigger and coassignment turns out to be

$$\begin{aligned}\text{TRG}_{\alpha, T'_\alpha(n)}(M'T') &= \text{TRG}_{\alpha, T_\alpha(i)}(M'T') \\ &= \text{ST}_\alpha(MT) + E_\alpha + \sum_{\substack{j=1 \\ j \neq i}}^{n-k} C_{\alpha, T_\alpha(j)}.\end{aligned}$$

The amount of variation of the trigger moment is

$$\begin{aligned}\text{TRG}_{\alpha, T_\alpha(i)}(M'T') - \text{TRG}_{\alpha, T_\alpha(i)}(MT) &= \sum_{\substack{j=1 \\ j \neq i}}^{n-k} C_{\alpha, T_\alpha(j)} - \sum_{j=1}^i C_{\alpha, T_\alpha(j)} \\ &= \sum_{j=i+1}^{n-k} C_{\alpha, T_\alpha(j)} - C_{\alpha, T_\alpha(i)}.\end{aligned}$$

(3) For child $T_\alpha(j)$, $i+1 \leq j \leq n$, the trigger moment after retrigger and coassignment becomes

$$\begin{aligned}\text{TRG}_{\alpha, T'_\alpha(j-1)}(M'T') &= \text{TRG}_{\alpha, T_\alpha(j)}(M'T') \\ &= \text{ST}_\alpha(MT) + E_\alpha + \sum_{\substack{m=1 \\ m \neq i \\ M(p) \neq M(T_p(m))}}^j C_{p, T_\alpha(m)}.\end{aligned}$$

The amount of variation of trigger moment is

$$\begin{aligned}\text{TRG}_{\alpha, T_\alpha(j)}(M'T') - \text{TRG}_{\alpha, T_\alpha(j)}(MT) &= \sum_{\substack{m=1 \\ m \neq i \\ M(p) \neq M(T_p(m))}}^j C_{p, T_\alpha(m)} - \sum_{\substack{m=1 \\ m \neq i \\ M(p) \neq M(T_p(m))}}^j C_{p, T_\alpha(m)} \\ &= -C_{\alpha, T_\alpha(i)}.\end{aligned}$$

Therefore, the total amount of variation of trigger moments is

$$\begin{aligned} & \left\{ \sum_{j=i+1}^{n-k} C_{\alpha, T_{\alpha}(j)} - C_{\alpha, T_{\alpha}(i)} \right\} - (n-i)C_{\alpha, T_{\alpha}(i)} \\ &= \sum_{j=i+1}^{n-k} C_{\alpha, T_{\alpha}(j)} - (n-i+1)C_{\alpha, T_{\alpha}(i)}. \quad \blacksquare \end{aligned}$$

If the i th triggered child is now coassigned and triggered last, it must wait for the k coassigned children to finish their execution on the coassigned processor. Therefore, after it is activated, it will wait for at least the amount of time $\sum_{j=n-k+1}^n E_{T_{\alpha}(j)}$ before being executed. Summing this delay and the total amount of variation of trigger moments in Theorem 2, we obtain the total cost variation for retriggering and coassigning $T_{\alpha}(i)$. If this total is negative (cost is diminished), which implies a reduction in communication overhead and in the balance of processor loads, then we will retrigger and coassign $T_{\alpha}(i)$.

In addition, if $C_{\alpha, T_{\alpha}(i)} \geq E_{T_{\alpha}(i)}$ for parent α and its child $T_{\alpha}(i)$, we will coassign $T_{\alpha}(i)$ and trigger it last. Thus, the load of processor $M(\alpha)$ will become $\text{TIME}_{M(\alpha)} + E_{T_{\alpha}(i)}$. On the contrary, if $T_{\alpha}(i)$ is not coassigned with its parent α , the load of $M(\alpha)$ turns out to be $\text{TIME}_{M(\alpha)} + C_{\alpha, T_{\alpha}(i)}$. As a consequence, after coassignment, the amount of load reduction for $M(\alpha)$ is $C_{\alpha, T_{\alpha}(i)} - E_{T_{\alpha}(i)}$. Moreover, the communication overhead for the last $n-i$ triggered children can also be lessened. Henceforth, retrigger and coassignment will be beneficial.

Summing up the above discussions, we have Heuristic 4 to delicately modify the initial transmission sequence.

HEURISTIC 4. Let there be an n/k fork where the parent is α . If $M(\alpha)$ is a zero-hop or one-hop processor of the other parents of $T_{\alpha}(i)$, and $C_{\alpha, T_{\alpha}(i)} \geq E_{T_{\alpha}(i)}$ or $(n-i+1)C_{\alpha, T_{\alpha}(i)} > \sum_{j=i+1}^{n-k} C_{\alpha, T_{\alpha}(j)} + \sum_{j=n-k+1}^n E_{T_{\alpha}(j)}$, where $1 \leq i \leq n-k$, then the i th triggered child will be coassigned with α and triggered last. Otherwise, its transmission sequence is not altered, and it will be assigned to a different processor from α 's.

The criterion that $M(\alpha)$ is a zero-hop processor or one-hop processor of the other parents of $T_{\alpha}(i)$ is to avoid multihop communication between the *coassigned* child and its parents. Thus, we will not incur too much communication overhead. The coassignment will be made from the first triggered child to the last one. If a child is coassigned with α in an n/k fork, it will become an $n/(k+1)$ fork.

Before describing Heuristic 5, let us define $\text{EF}_{\alpha}(MT) = \text{ACT}_{\alpha}(MT) + E_{\alpha}$ to be the estimated finish moment of module α associated with the task assign-

ment MT . If there are several modules activated concurrently, we have to decide which one will first determine the transmission sequence and coassignment of its children. From the viewpoint of load balance, we require to diminish the load of the processor on which the activated module with the largest EF value resides. However, the larger EF value for a module may be caused by its larger level, where by *level* we mean the length of the longest path from source modules to the module. Such a condition may result in the module with smaller EF value triggering and coassigning its children earlier than the one with larger EF value. Therefore, a worse task assignment may be derived.

Modules are leveled from source modules to sink modules. Initially, the levels of all source modules are set to zero. Then, the level l_β of module β is defined as

$$l_\beta = \max_{\alpha \in F_\beta} l_\alpha + 1.$$

As mentioned previously, F_β denotes the set of β 's parents.

Considering both level and EF value, we have Heuristic 5 to determine the priority of coassignment and transmission-sequence decisions when several modules are activated concurrently.

HEURISTIC 5. The activated module with the smallest level and the largest estimated finish moment first determines the coassignment and transmission sequence of its children. That is, the activated modules in the wait list are sorted in the nondecreasing order of level and the nonincreasing order of estimated finish moment. In addition, the former takes precedence over the latter.

C. MODULE ASSIGNMENT

Recall that, in Heuristic 4, some children are coassigned with their parents and others are not. We attempt to assign the uncoassigned child to the one-hop processor of its parents which leads to the minimum start moment of the child and the minimum elapsed time (load) of the processor. The use of the minimum start moment of a module implies that the processors assigned to its children and descendants will have smaller elapsed times. For a module, all of the one-hop processors of its parents are called its *available processors* for short.

From Equations (2) and (3), we conclude that a possible minimum start moment of an unassigned module is its activation moment. The set of α 's available processors whose elapsed times are larger than the activation moment of module α is called the L set of α , and its complement the S set of α .

In order to attain the minimum start moment of α , we require to assign α to a member of its S set. Thus both α 's start moment and the processor's elapsed time are just the activation moment of α . However, from Equation (5), we know that there exists idle time for this processor. If α is assigned to the processor in the S set whose elapsed time is closest to α 's activation moment, the idle time of this processor will be the minimum one among all processors in the S set.

If the S set of α is empty and then we choose the processor with the least load in the L set to be α 's assigned processor, the elapsed time of this processor will become α 's start moment. Hence, this start moment will be the minimum one for α . Furthermore, since α is executed on the processor with the least load, load balance can be achieved. Hence, we have Heuristic 6 to sophisticatedly allocate an activated module to the best-fit processor.

HEURISTIC 6. Let there be an activated module which is not coassigned with any parent. If there exist available processors whose elapsed times are not larger than the module's activation moment, we assign it to the one with the largest elapsed time. If there is no such processor, we assign it to the available processor with the smallest elapsed time.

If, unfortunately, there is no available processor for module α , then α can only be assigned to a multihop processor of its parents. We adopt static routing to determine the paths of message transmission between its parents' processors and the multihop processor, since static routing is an effective routing method and one of the most widely used [18]. In practice, under this condition, we first select the processor with the least elapsed time to be α 's assigned processor. The, for each of α 's parents, we apply Algorithm 1 to compute the trigger moments of the parent of α for all possible routes provided by the static routing. After doing that, we choose the route resulting in the least trigger moment to be the path for message transmission between the parent's processor and the multihop processor.

D. REASSIGNMENT

If task partitioning does not successfully exploit parallelism and minimize communication overhead, the minimum task turnaround time will be very close to the sum of execution times of all modules (called the *total execution time*). Therefore, the task turnaround time obtained by the aforementioned heuristics may exceed the total execution time. Although this may rarely happen in a reliable task partitioning, we still cannot take the chance of the attained task turnaround time exceeding the total execution time due to a bad task partitioning.

To handle such an abnormal condition, we check whether or not the attained task turnaround time is larger than the total execution time. If so, we reassign all modules to a single processor. This seems a simple policy, but is an effective one if combined with the proposed six heuristics. Using the approach of iterative improvement, we can also reassign modules and alter transmission sequences to reduce the task turnaround time, but that is time-consuming. In consequence, we do not suggest it in this paper.

IV. ILLUSTRATIVE EXAMPLE AND EXPERIMENTAL RESULTS

The algorithm using the proposed heuristics and reassignment to solve the task assignment problem in distributed computing systems is presented in the appendix. An example is now given to illustrate it. The illustrative distributed computing system is composed of three processors, and the task is partitioned into seven modules with precedence constraints. They are modeled as an undirected graph and a DAG graph as depicted in Figures 3 and 4 respectively. We want to assign the seven modules to the three processors in order to obtain a near-minimum task turnaround time. Tables 1 and 2 show the execution time of each module and the intermodule communication time of each adjacent module respectively. Also, Table 3 lists the route between multihop processors.

Fig. 3.

Fig. 4.

TABLE 1
Execution Time

Module	Time
<i>A</i>	10
<i>B</i>	60
<i>C</i>	80
<i>D</i>	40
<i>E</i>	50
<i>F</i>	45
<i>G</i>	20

TABLE 2
Intermodule Communication Time

Module	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>	<i>F</i>	<i>G</i>
<i>A</i>	∞	30	40	10	∞	∞	∞
<i>B</i>	∞	∞	∞	∞	75	∞	∞
<i>C</i>	∞	∞	∞	∞	10	30	∞
<i>D</i>	∞	∞	∞	∞	∞	60	∞
<i>E</i>	∞	∞	∞	∞	∞	∞	15
<i>F</i>	∞	∞	∞	∞	∞	∞	10
<i>G</i>	∞	∞	∞	∞	∞	∞	∞

The task assignment *MT* obtained by the proposed algorithm for the illustrative example is

$$\begin{aligned}
 M(A) &= 1, & T_A &= \{B, D, C\}, \\
 M(B) &= 2, & T_B &= \{E\}, \\
 M(C) &= 1, & T_C &= \{E, F\}, \\
 M(D) &= 3, & T_D &= \{F\}, \\
 M(E) &= 2, & T_E &= \{G\}, \\
 M(F) &= 1, & T_F &= \{G\}, \\
 M(G) &= 1, & T_G &= \{ \},
 \end{aligned}$$

where the *M* denotes a module and its assigned processor, and the *T* denotes the list of transmission sequence of its children. Figure 5 plots the timing

TABLE 3
Routes between Multihop Processors

Processor		Route
From	To	
2	3	2 → 1 → 3
3	2	3 → 1 → 2

diagram of the obtained task assignment. We achieve a near-minimum task turnaround time whose value is 225 time units. Comparing it with the minimum one, whose value is 215 time units, we see that we incur only 4.7% turnaround-time deviation.

A heuristic task-assignment simulator has been developed on a VAX 8200 by using the language c. A task-assignment simulator achieving the optimal solution has also been developed. Tens of experiments have been made for each of three EC ratios (average execution time divided by average communication time): 8, 4, and 1. In these experiments, execution times and communication times are generated by a random-number generator with average values statistically yielding some EC ratio. Besides, the topologies of the distributed computing systems and tasks are generated randomly, and every vertex is linked to at least one of the other vertices. Several routes are also provided for every pair of multihop processors.

The percentage of optimal solutions generated by the heuristic algorithm is called the *hit ratio*, and the percentage deviation of its obtained task turnaround time from the minimum one is called the *deviation* for short. From

Fig. 5.

TABLE 4
Experimental Results

EC ratio	8	4	1
Average deviation	6.7	7.8	4.5
Hit ratio	45	50	74
Re ^a	8	16	36
OE ^b	4	8	20

^a(Number of reassignments/number of experiments) $\times$ 100%.

^b(Number of optimal solutions using reassignment/number of experiments) $\times$ 100%.

the experimental results listed in Table 4, we have the following interesting results:

(1) The average deviations are very small for all EC ratios. Also, the probabilities of the worst cases, whose deviations are about 20%, are below 2% for all EC ratios.

(2) The average hit ratios are as high as 45%, 50% and 74% for EC ratios 8, 4, and 1 respectively.

(3) The reassignment plays a more important role for smaller EC ratios than larger ones. For the large EC ratios 8 and 4, only 4/45 and 8/50 of the optimal solutions are attained by using the reassignment. However, if we only use the proposed six heuristics, we can still obtain a high percentage of optimal solutions: 41%, 42%, and 54% for EC ratios 8, 4, and 1 respectively.

In the proposed algorithm, Heuristic 4 and routing are the dominant parts of the time complexity, but we know that the latter rarely occurs. We have used an $O(m^2p)$ algorithm to determine transmission sequences and coassignments in Heuristic 4, and an $O(m^2pl)$ algorithm to decide the most suitable processor and path for routing between multihop processors, where m denotes the number of modules, p the number of processors, and l the maximum number of routes between multihop processors. Therefore, the proposed algorithm provides polynomial time complexity to solve the problem of task assignment in distributed computing systems.

V. CONCLUSION

The task-assignment problem dealt with in this paper takes precedence constraint into consideration. In addition, the cost function to measure the performance of task assignment includes not only the execution time and the communication time, but also the idle time. Based on the module assignment

and transmission sequence, task turnaround time is successfully defined by using the concepts of trigger moment, activation moment, and start moment.

It is known that the search for an optimal task assignment with precedence constraint is NP-complete in the strong sense [10, 15, 16]. Six heuristics combined with an effective reassignment are proposed to effectively solve the task assignment problem. Though it is a heuristic approach, a near-optimal or even the optimal solution can be attained. If task partitioning successfully exploits parallelism and minimizes communication overhead, we can drop reassignment but still achieve satisfactory solutions. The small deviations and the high hit ratios of experimental results show that the proposed approach can effectively solve the task-assignment approximation problem in distributed computing systems.

APPENDIX

Algorithm 2.

- (1) Initialize the elapsed time of all processors to zero;
Initialize the levels, trigger moments, activation moments and start moments of all source modules to zero;
- (2) Levelize modules;
- (3) Compute the estimated finish moments of source modules and insert them into the wait list according to Heuristic 5;
- (4) WHILE (the wait list is not empty)
- (5) BEGIN
- (6) Move the first element of the wait list to α ;
- (7) IF (α is not coassigned with any parent)
- (8) THEN use Heuristic 6 to assign α to the best-fit processor;
- (9) IF ($M(\alpha)$ is a multihop processor of its parents)
- (10) THEN recompute the trigger moments, activation moments, and start moments of the currently assigned modules;
- (11) Compute α 's start moment and finish moment, and update the elapsed time of $M(\alpha)$ with both terms;
- (12) Sort n children of α according to Heuristic 3;
- (13) Use Heuristic 1 to determine whether execution time or communication time will be the metric of the initial transmission-sequence decision;
- (14) IF (the execution time is adopted as the metric)
- (15) THEN re-sort children according to Heuristic 2;
- (16) FOR $i = 1$ TO n DO
- (17) IF ($T_\alpha(i)$ is not assigned)

```

(18) THEN use Heuristic 4 to determine whether  $T_\alpha(i)$  is coassigned with
       $\alpha$  and triggered last;
(19) ELSE IF ( $M(T_\alpha(i))$  is the same as  $M(\alpha)$ )
(20) THEN trigger  $T_\alpha(i)$  last;
/* The transmission sequence of  $\alpha$  to its children is already changed. */
(21) FOR  $i = 1$  TO  $n$  DO
(22) BEGIN
(23) IF ( $T_\alpha(i)$  is coassigned with  $\alpha$ )
(24) THEN  $\text{TRG}_{\alpha, T_\alpha(i)} = \text{TIME}_{M(\alpha)}$ ;
(25) ELSE  $\text{TIME}_{M(\alpha)} = \text{TRG}_{\alpha, T_\alpha(i)} = \text{TIME}_{M(\alpha)} + C_{\alpha, T_\alpha(i)}$ ;
(26) IF ( $T_\alpha(i)$  is activated)
(27) THEN compute its activation moment and estimated finish moment,
      and insert it into the wait list according to Heuristic 5;
(28) ENDFOR;
(29) ENDWHILE;
(30) Use Equation (6) to compute the task turnaround time;
(31) IF (the task turnaround time is larger than the total execution time)
(32) THEN do reassignment;
(33) END of Algorithm 2.

```

REFERENCES

1. S. H. Bokhari, Dual processor scheduling with dynamic reassignment, *IEEE Trans. Software Engrg.* SE-5:341–349 (July 1979).
2. H. S. Stone, Multiprocessor scheduling with the aid of network flow algorithms, *IEEE Trans. Software Engrg.* SE-3(6):85–93 (Jan. 1977).
3. H. S. Stone and S. H. Bokhari, Control of distributed processes, *Computer* 11:97–106 (July 1978).
4. W. W. Chu, L. J. Holloway, M. T. Lan, and K. Efe, Task allocation in distributed data processing, *Computer* 13:57–69 (Nov. 1980).
5. V. M. Lo, Task assignment to minimize completion time, in *Proceedings of the 5th International Conference on Distributed Computing Systems*, IEEE, Denver, May 1985, pp. 329–336.
6. P. Y. R. Ma, E. Y. S. Lee, and M. Tsuchiya, A task allocation model for distributed computing systems, *IEEE Trans. Comput.* C-31(1):41–47 (Jan. 1982).
7. K. Efe, Heuristic models of task assignment scheduling in distributed systems, *Computer* 15:50–56 (June 1982).
8. V. Lo and J. W. S. Liu, Task assignment in distributed multiprocessor system, in *Proceedings of 1981 International Conference on Parallel Processing*, Columbus, Ohio, Aug. 1981, pp. 358–360.
9. C. C. Shen and W. H. Tsai, A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion, *IEEE Trans. Comput.* C-34(3):197–203 (Mar. 1985).
10. H. Kasahara and S. Narita, Practical multiprocessor scheduling algorithms for efficient parallel processing, *IEEE Trans. Comput.* C-33(11):1023–1029 (Nov. 1984).

11. A. K. Ezzat, R. D. Bergeron, and J. L. Pokoski, Task allocation heuristics for distributed computing systems, in *Proceedings of the 6th International Conference on Distributed Computing Systems*, IEEE, Cambridge, Mass., 1986, pp. 337–346.
12. J. B. Sinclair, Efficient computation of optimal assignments for distributed tasks, *J. Parallel and Distributed Comput.* 4:342–362 (1987).
13. J. Sheild, Partitioning concurrent VLSI simulation programs onto a multiprocessor by simulated annealing, *IEE Proc.* 134(Pt. E, No. 1):24–30 (Jan. 1987).
14. S. H. Bokhari, A. shortest tree algorithm for optimal assignments across space and time in a distributed processor system, *IEEE Trans. Software Engrg.* SE-7(6):583–589 (Nov. 1981).
15. S. Sahni and T. Gonzalez, P-complete approximation problems, *J. Assoc. Comput. Mach.* 23(7):555–565 (1976).
16. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.
17. W. T. Chen and J. P. Sheu, Task assignment in loosely-coupled distributed computing systems, *J. Chinese Inst. Engrs.* 10(6):721–726 (1987).
18. A. S. Tanenbaum, *Computer Networks*, Prentice-Hall, Englewood Cliffs, N.J., 1981.
19. W. W. Chu and L. M. Lan, Task allocation and precedence relations for distributed real-time system, *IEEE Trans. Comput.* C-36(6):667–679 (June 1987).

Received 4 October 1989; revised 11 November 1989, 21 December 1989