UNIVERSITY AT STONY BROOK

CEAS Technical Report 697

A Multi-Job Load Sharing Strategy for Divisible Jobs on Bus Networks

J. Sohn and T.G. Robertazzi

Aug. 15, 1994

A Multi-Job Load Sharing Strategy for Divisible Jobs on Bus Networks

Jeeho Sohn and Thomas G. Robertazzi, *Senior Member, IEEE*

Dept. of Electrical Engineering,

SUNY at Stony Brook,

Stony Brook, N.Y. 11794

A Multi-Job Load Sharing Strategy for Divisible Jobs on Bus Networks

Prof. Thomas G. Robertazzi

Dept. of Electrical Engineering

SUNY at Stony Brook

Stony Brook, N.Y. 11794

516-632-8412

Abstract

In this paper, a load sharing problem involving the optimal load allocation of very long linear data files over N processors interconnected in a bus-oriented network is investigated. Two distinct types of bus networks are examined in the case when there is more than one such outstanding divisible job in the network. These are a network with load origination at a control processor and a network with load origination at a computing processor. In both cases, it is assumed that each processor is equipped with a front-end processor for communications off-loading. It is found that a *multi-job* scheme outperforms a single-job scheme in terms of the total solution finish time. Closed form solutions and simple recursive algorithms for the determination of the optimal load allocation are also presented.

$\hat{\alpha}_i^m$: Greek letter alpha with hat,

subscript English letter small i, superscript English letter small m

α_i^m : Greek letter alpha,

subscript English letter small i, superscript English letter small m

Z : English letter capital Z

w_i : English letter small w,

subscript English letter small i

T_{cm}^m : English letter capital T,

subscript English letter small c and m, superscript English letter small m

T_{cp}^m : English letter capital T,

subscript English letter small c and p, superscript English letter small m

T_f^m : English letter capital T,

subscript English letter small f, superscript English letter small m

k_i: English letter small k,

subscript English letter small i

Γ_i: Greek letter capital Gamma,

subscript English letter small i

u_n^m: English letter small u,

subscript English letter small n, superscript English letter small m

v_n^m: English letter small v,

subscript English letter small n, superscript English letter small m

1. INTRODUCTION

In recent years, there has been of great deal of interest in distributed sensor networks [20]. In distributed sensor networks, measurements are made by spatially distinct sensors. The data is, then, broadcast to a site where the spatially disparate readings are fused so that meaningful decisions can be made regarding these measurements. One major issue for distributed sensor networks is the trade-off between communication and computation [12]. That is, the decision of how much time should be spent to communicate the measurements and how much time should be spent to process (compute) the measurements becomes an important problem.

Related to the distributed sensor network problem are a number of papers which deal with scheduling and load sharing in multiprocessors [9, 19]. However most work assumes that a job can be assigned to at most a single processor. Only recently has there been interest in multiprocessor scheduling with jobs that need to be assigned to more than one processor [8, 13, 21].

Recently there has been work on a load sharing problem involving a *divisible* job. A divisible job is a job that can be arbitrarily partitioned in a linear fashion among a number of processors. Applications include the processing of very long linear data files as in signal and image processing and Kalman filtering.

In [10], recursive expressions for calculating the optimal load allocation for linear daisy chains of processors were presented. This is based on the simplifying premise that for an optimal allocation of load, all processors must stop processing at the same time. Intuitively, this is because otherwise some processors would be idle while others were still busy. Analogous solutions have been developed for tree networks [11] and bus networks [1, 2]. Asymptotic solutions for systems with large or even an infinite number of processors and limitations in performance when adding processors appear in [3, 15]. Closed form solutions were presented

in [4] for bus and tree architectures where processor and link speeds are homogeneous. In [17], the concept of an equivalent processor that behaves identically to a collection of processors in the context of a linear daisy chain of processors and a proof that, for a linear daisy chain of processors load sharing a divisible job, the optimal solution involves all processors stopping at the same time are introduced. An analytic proof for bus networks that for a minimal time solution all processors must stop computing at the same time appears in [18]. Previous proofs were heuristic. The equivalence of first distributing load either to the left or to the right from a point in the interior of a linear daisy chain is demonstrated in [14]. Optimal sequences of load distribution in tree networks are described in [5, 16]. A new load distribution strategy for tree networks [6] and linear daisy chains [7] has also been discussed.

All this previous work concentrated on investigating more efficient ways of distributing load to minimize the total solution time under the premise that there is only one job present in the network. However most practical computer systems operate in an environment where multiple jobs may be submitted. An intriguing question is how to efficiently handle the submission of multiple divisible jobs in a distributed network. Under a naive, *single-job scheme*, a distributed network sequentially processes one job at a time. In this paper we propose a more sophisticated *multi-job scheme* for bus networks that exploits the special structure of a divisible job on a bus network to yield a smaller finish (solution) time than the single-job scheme.

This paper is organized as follows: Closed form solutions and simple recursive algorithms for the determination of the optimal load allocation for two types of bus networks – a network with load origination at a control processor and a network with load origination at a computing processor – are presented in section 2 and 3, respectively. In section 4, a performance evaluation is presented which compares the single-job scheme with the multi-job scheme. Finally the conclusion appears in section 5.

2. ARCHITECTURE 1:

LOAD ORIGINATION AT A CONTROL PROCESSOR

Consider first the case where the network model consists of a queueing system for incoming jobs, a control processor for distributing the processing load, and N processors attached to a linear bus as in Fig. 1. New arriving jobs enter the queueing system and wait there for service. In this paper, the queueing system is assumed to be large enough in size so that there are no *turned away*, *lost* or *blocked* jobs. It is also assumed that the service discipline of the queueing system is *first-in first-out* (FIFO). Under the supervision of the control processor, one of the waiting jobs in the queue can be delivered to the processor. The control processor distributes the processing load among the N processors interconnected through a bus type communication medium in order to obtain the benefits of parallel processing. The control processor does no processing itself. Each processor is equipped with a front-end processor to make it possible to compute and communicate simultaneously. Put another way, the front-end processor handles communication duties and allow the main processor to concentrate on computation. Each processor may have a different computing speed.

The following notations will be used throughout this paper:

$\hat{\alpha}_i^m$: The fraction of the entire processing load of the mth job that is assigned to the ith processor in *the single-job scheme.*

α_i^m : The fraction of the entire processing load of the mth job that is assigned to the ith processor in *the multi-job scheme.*

Z : The inverse of the channel speed of the bus.

w_i : The inverse of the computing speed of the ith processor.

T_{cm}^m : The time that it takes to transmit the entire set of measurement data of the mth job over the channel when $Z = 1$.

T_{cp}^m : The time that it takes for each processor to process (compute) the entire load of the mth job when $w_i = 1$.

T_f^m : The finish time of the entire processing load of the mth job.

The timing diagram for the bus network with load origination at a control processor in the single-job scheme and the multi-job scheme are depicted in Fig. 2 and Fig. 3, respectively. In these timing diagrams communication time appears above the axis and computation time appears below the axis. It is assumed that at the time of origin, the network with queueing system has already received more than one job from outside so that multiple jobs are in the queue and all the processors are idle, and the control processor starts distributing the first available job's load to the other processors through the bus. In Fig. 2, it takes a time $\hat{\alpha}_1^1 Z T_{cm}^1$ for the control processor to transmit the first fraction of the first job's load to the first processor and a time $\hat{\alpha}_2^1 Z T_{cm}^1$ to transmit the second fraction of the first job's load to the second processor, and so on. Then after the first processor completed receiving the load from the control processor which is an amount of $\hat{\alpha}_1^1$ of the entire load of the first job, it can start computing immediately and it will take a time of $\hat{\alpha}_1^1 w_1 T_{cp}^1$ to finish. The second processor also completes receiving the load from the control processor at time $(\hat{\alpha}_1^1 + \hat{\alpha}_2^1) Z T_{cm}^1$ and it will start computing for a duration of $\hat{\alpha}_2^1 w_2 T_{cp}^1$ of time. This procedure continues until the last processor. For optimality, all the processors must finish computing at the same time. Intuitively this is because otherwise the solution time could be improved by transferring load from busy processor to idle one [17, 18]. The single-job scheme is the scheme in which the load distribution of the next job is started only after the completion of the current job's computation (T_f^1). Therefore, starting with the second job, the time that each processor waits for its processing load to be received is wasted. That is, the durations

$\hat{\alpha}_1^m ZT_{cm}^m, (\hat{\alpha}_1^m + \hat{\alpha}_2^m)ZT_{cm}^m, \ldots, (\hat{\alpha}_1^m + \hat{\alpha}_2^m + \cdots + \hat{\alpha}_N^m)ZT_{cm}^m$ for $m = 2, 3, \ldots$, are wasted in processor 1, processor 2, ..., processor N, respectively.

The proposed multi-job scheme, on the other hand, is based on the idea that one wants to reduce the wasted time between two consecutive jobs. Therefore, in this scheme, the control processor dose not need to wait until the finish time of the previous job's processing and can start distributing the load immediately after it finishes the transmission of the previous job's load so that those processors which have completed receiving their processing load prior to the completion of the previous job's computation (T_f^1) can start their computation immediately after T_f^1. Clearly, one can significantly reduce the overall processing time starting with the second job in this multi-job scheme. Furthermore, this multi-job scheme is optimum. The reasons why it is optimum will be discussed later.

Now, the closed-form solutions to find $\hat{\alpha}_i^m$ for the optimal load allocation for each processor in the single-job scheme which appears in [18] will be briefly reviewed. Following this, the recursive algorithm to find α_i^m for $m \geq 2$ for the optimal load allocation in the multi-job scheme will be examined.

2.1. *Closed-Form Solution to Find $\hat{\alpha}_i^m$ in the Single-Job Scheme*

Since distributing the processing load is done in the same fashion for every job in the single-job scheme and for the first job in the multi-job scheme, the closed-form solutions to find $\hat{\alpha}_i^m$ and α_i^1 are the same in both cases. The case of finding $\hat{\alpha}_i^m$ for the mth job in the single-job scheme will thus solely be reviewed.

As proved in [18], the minimum time solution occurs when all processors finish their computation at the same time. Based on that fact, one can set up the following set of equations from Fig. 2. This equates the computation time of the ith processor to the transmission plus computation time of the $i+1$st processor. Although Fig. 2 shows the timing diagram only

for the first job and the second job this section will demonstrate the closed-form solution to find the optimum fraction $\hat{\alpha}_i^m$ for the general mth job.

$$\hat{\alpha}_i^m w_i T_{cp}^m = \hat{\alpha}_{i+1}^m Z T_{cm}^m + \hat{\alpha}_{i+1}^m w_{i+1} T_{cp}^m \qquad i = 1, 2, \ldots, N-1 \tag{1}$$

This equation can be solved as

$$\begin{aligned} \hat{\alpha}_{i+1}^m &= \frac{w_i T_{cp}^m}{Z T_{cm}^m + w_{i+1} T_{cp}^m} \hat{\alpha}_i^m \\ &= k_i \hat{\alpha}_i^m \\ &= k_i k_{i-1} \cdots k_1 \cdot \hat{\alpha}_1^m \qquad i = 1, 2, \ldots, N-1 \end{aligned} \tag{2}$$

where

$$k_i = \frac{\hat{\alpha}_{i+1}^m}{\hat{\alpha}_i^m} = \frac{w_i T_{cp}^m}{Z T_{cm}^m + w_{i+1} T_{cp}^m} \qquad i = 1, 2, \ldots, N-1 \tag{3}$$

Here k_i can be directly found from the system parameters (w_i, Z) and the size of the mth job (T_{cm}^m, T_{cp}^m). Since the fractions of the total processing load should sum to one, $\hat{\alpha}_1^m$ can be obtained by the normalization equation.

$$\sum_{i=1}^{N} \hat{\alpha}_i^m = [\, 1 + \sum_{i=1}^{N-1} (\prod_{j=1}^{i} k_j) \,]\, \hat{\alpha}_1^m = 1 \tag{4}$$

Once $\hat{\alpha}_1^m$ is found, one can easily calculate the rest of the load allocation fractions $(\hat{\alpha}_2^m, \hat{\alpha}_3^m, \ldots, \hat{\alpha}_N^m)$ by using Eq.(2). Note that the single-job scheme is optimal if only a single job is to be solved. When multiple jobs are involved it is not optimal, as the following sections will demonstrate.

2.2. *Algorithm to Find α_i^m in the Multi-Job Scheme: When $T_f^{m-1} \geq Z T_{cm}^{m-1} + Z T_{cm}^m$ for $m \geq 2$*

This section will consider the case that the transmission of the mth job's processing load is started immediately after the control processor finishes the transmission of the $(m-1)$st job's processing load and is finished before the completion of the $(m-1)$st job's computation

(T_f^{m-1}) as shown in Fig. 4. In other words, the location of T_f^{m-1} is placed after the time that the transmission of the mth job's processing load is finished. In this case, the algorithm to find the α_i^m is very simple. Note that Fig. 4 shows the timing diagram only for the first two jobs and the transmission time for each fraction of load is abbreviated to only its corresponding fraction (e.g., $\alpha_1^1 Z T_{cm}^1$ just as α_1^1). For our purposes one may think of the first job as the $(m-1)$st job and the second job as the mth job.

Since all the processors have received their processing load prior to the finish time of the previous job (T_f^{m-1}) they can simultaneously start their computation immediately after T_f^{m-1} and finish their computation at the same time for a minimal solution time. Therefore, the processing time in each processor is the same. That is,

$$\alpha_1^m w_1 T_{cp}^m = \alpha_2^m w_2 T_{cp}^m = \alpha_3^m w_3 T_{cp}^m = \cdots = \alpha_N^m w_N T_{cp}^m \tag{5}$$

and those equations can be further expressed as a function of α_1^m.

$$\alpha_i^m = \frac{w_1}{w_i} \alpha_1^m \qquad i = 1, 2, \ldots, N \tag{6}$$

As the normalization equation states that the sum of the fractions of the total processing load is one, α_1^m can be obtained.

$$\sum_{i=1}^{N} \alpha_i^m = \sum_{i=1}^{N} \frac{w_1}{w_i} \alpha_1^m = 1 \tag{7}$$

Therefore, one can derive a simple closed-form solution.

$$\alpha_1^m = (\sum_{i=1}^{N} \frac{w_1}{w_i})^{-1} \tag{8}$$

$$\alpha_i^m = \frac{w_1}{w_i} \alpha_1^m \qquad i = 1, 2, \ldots, N \tag{9}$$

Note that since there is no wasted time between the consecutive jobs in every processor, the total required processing time for the mth job ($T_f^m - T_f^{m-1}$) is minimized compared to any other cases.

2.3. *Algorithm to Find α_i^m in the Multi-Job Scheme: When $T_f^{m-1} < ZT_{cm}^{m-1} + ZT_{cm}^m$ for $m \geq 2$*

This subsection will consider the case when T_f^{m-1} is located somewhere during the transmission period of the mth job's load. That is, some of the processors which have received their processing load before T_f^{m-1} can start their computation immediately after T_f^{m-1} while the other processors which did not receive their processing load yet have to wait some period of time for their processing load to be delivered. In Fig. 5, again one can think of the first job as the $(m-1)$st job and the second job as the mth job. The processing finish time of the $(m-1)$st job's load (T_f^{m-1}) is in the transmission of the nth fraction of the mth job's load. Let Γ_i denote the time interval between T_f^{m-1} and the time that the ith processor receives its processing load for the mth job. Thus processor 1 to processor n-1 can start their computation immediately after T_f^{m-1} while processor n, processor n+1, ..., processor N have to wait $\Gamma_n, \Gamma_{n+1}, \ldots, \Gamma_N$ amounts of time to receive their processing load and start their computation at time $T_f^{m-1} + \Gamma_n, T_f^{m-1} + \Gamma_{n+1}, \ldots, T_f^{m-1} + \Gamma_N$, respectively.

Now let us derive the algorithm to find α_i^m which minimizes the total solution time. We will assume that all processors must stop at the same time. The processing time of processor 1 to processor n-1 is the same since they simultaneously start their computation and finish it at the same time.

$$\alpha_1^m w_1 T_{cp}^m = \alpha_2^m w_2 T_{cp}^m = \cdots = \alpha_{n-1}^m w_{n-1} T_{cp}^m \tag{10}$$

Therefore,

$$\alpha_i^m = \frac{w_1}{w_i} \alpha_1^m \qquad i = 1, 2, \ldots, n-1 \tag{11}$$

The next step is to find an expression for Γ_i, which is the time interval between T_f^{m-1}

and the time the ith processor receives its processing load.

$$\Gamma_i = (\alpha_1^m + \alpha_2^m + \cdots + \alpha_i^m) Z T_{cm}^m - \alpha_N^{m-1} w_N T_{cp}^{m-1} \qquad i = n, n+1, \ldots, N \tag{12}$$

The processing time of the ith processor to compute its processing load is then

$$\begin{aligned} \alpha_i^m w_i T_{cp}^m &= \alpha_1^m w_1 T_{cp}^m - \Gamma_i \\ &= \alpha_1^m w_1 T_{cp}^m - (\sum_{k=1}^{i} \alpha_k^m) Z T_{cm}^m + \alpha_N^{m-1} w_N T_{cp}^{m-1} \qquad i = n, n+1, \ldots, N \end{aligned} \tag{13}$$

Now consider the processing time taken by the nth processor to compute its processing load, i.e., the processing time when $i = n$.

$$\alpha_n^m w_n T_{cp}^m = \alpha_1^m w_1 T_{cp}^m - (\alpha_1^m + \alpha_2^m + \cdots + \alpha_{n-1}^m + \alpha_n^m) Z T_{cm}^m + \alpha_N^{m-1} w_N T_{cp}^{m-1}$$

Here α_n^m can be expressed by rearranging, as a function of α_1^m since $\alpha_2^m, \alpha_3^m, \ldots, \alpha_{n-1}^m$ are function of α_1^m as found previously.

$$\begin{aligned} \alpha_n^m &= \frac{w_1 T_{cp}^m - (\sum_{i=1}^{n-1} \frac{w_1}{w_i}) Z T_{cm}^m}{Z T_{cm}^m + w_n T_{cp}^m} \alpha_1^m + \frac{\alpha_N^{m-1} w_N T_{cp}^{m-1}}{Z T_{cm}^m + w_n T_{cp}^m} \\ &= u_n^m \alpha_1^m + v_n^m \end{aligned} \tag{14}$$

Here u_n^m and v_n^m are constants which are functions of system parameters (Z, w_i), the computation size of the $(m-1)$st job (T_{cp}^{m-1}), the size of the mth job (T_{cp}^m, T_{cm}^m), and (α_N^{m-1}). [1]

$$u_n^m = \frac{w_1 T_{cp}^m - (\sum_{i=1}^{n-1} \frac{w_1}{w_i}) Z T_{cm}^m}{Z T_{cm}^m + w_n T_{cp}^m} \tag{15}$$

$$v_n^m = \frac{\alpha_N^{m-1} w_N T_{cp}^{m-1}}{Z T_{cm}^m + w_n T_{cp}^m} \tag{16}$$

Let us consider the case for the next processor, i.e., when $i = n+1$.

$$\begin{aligned} \alpha_{n+1}^m w_{n+1} T_{cp}^m &= \alpha_1^m w_1 T_{cp}^m - \Gamma_{n+1} \\ &= \alpha_1^m w_1 T_{cp}^m - (\alpha_1^m + \alpha_2^m + \cdots + \alpha_{n-1}^m + \alpha_n^m + \alpha_{n+1}^m) Z T_{cm}^m + \alpha_N^{m-1} w_N T_{cp}^{m-1} \end{aligned}$$

[1] Here α_N^{m-1} is assumed to be a known value because this is the fraction found when the $(m-1)$st job is distributed.

Again α_{n+1}^m can be expressed as a function of α_1^m.

$$\begin{aligned}\alpha_{n+1}^m &= \frac{w_1 T_{cp}^m - (\sum_{i=1}^{n-1} \frac{w_1}{w_i} + u_n^m) Z T_{cm}^m}{Z T_{cm}^m + w_{n+1} T_{cp}^m} \alpha_1^m + \frac{\alpha_N^{m-1} w_N T_{cp}^{m-1} - v_n^m Z T_{cm}^m}{Z T_{cm}^m + w_{n+1} T_{cp}^m} \\ &= u_{n+1}^m \alpha_1^m + v_{n+1}^m\end{aligned}$$

where u_{n+1}^m and v_{n+1}^m are also constants since u_n^m and v_n^m have been found previously.

One can see that this procedure can be continued up to the case where $i = N$. Then every fraction that the originating processor should calculate to achieve the minimum solution time is found as a function of α_1^m.

Once all α_i^m have been found, α_1^m can then be calculated from the normalization equation.

$$\begin{aligned}1 &= \sum_{i=1}^{n-1} \alpha_i^m + \sum_{i=n}^{N} \alpha_i^m \\ &= (\sum_{i=1}^{n-1} \frac{w_1}{w_i}) \alpha_1^m + \sum_{i=n}^{N} (u_i^m \alpha_1^m + v_i^m) \\ &= (\sum_{i=1}^{n-1} \frac{w_1}{w_i} + \sum_{i=n}^{N} u_i^m) \alpha_1^m + \sum_{i=n}^{N} v_i^m \end{aligned} \tag{17}$$

As a summary of this section, let us rephrase the algorithm to find α_i^m when the case $T_f^{m-1} < Z T_{cm}^{m-1} + Z T_{cm}^m$ for $m \geq 2$.

(1)
$$u_i^m = \frac{w_1 T_{cp}^m - (\sum_{k=1}^{n-1} \frac{w_1}{w_k} + \sum_{k=n}^{i-1} u_k^m) Z T_{cm}^m}{Z T_{cm}^m + w_i T_{cp}^m} \qquad i = n, n+1, \ldots, N \tag{18}$$

$$v_i^m = \frac{\alpha_N^{m-1} w_N T_{cp}^{m-1} - (\sum_{k=n}^{i-1} v_k^m) Z T_{cm}^m}{Z T_{cm}^m + w_i T_{cp}^m} \qquad i = n, n+1, \ldots, N \tag{19}$$

(2)
$$\alpha_1^m = \frac{1 - \sum_{i=n}^{N} v_i^m}{\sum_{i=1}^{n-1} \frac{w_1}{w_i} + \sum_{i=n}^{N} u_i^m} \tag{20}$$

(3)
$$\alpha_i^m = \begin{cases} \frac{w_1}{w_i} \alpha_1^m & i = 1, 2, \ldots, n-1 \\ u_i^m \alpha_1^m + v_i^m & i = n, n+1, \ldots, N \end{cases} \tag{21}$$

where it is defined that $\sum_{i=n}^{n-1} u_i^m \text{or } v_i^m = 0$.

2.4. *When the position of T_f^{m-1} is changed due to the new values of α_i^m*

Reading along so far, one should understand that an unrealizable situation has been described as one can ask how can one determine in which fraction of transmission T_f^{m-1} is located even before the actual values of α_i^m are calculated. Actually it may seem impossible to determine the location of T_f^{m-1} because to do that, all values of α_i^m should be calculated first and to calculate the values of α_i^m, the location of T_f^{m-1} should be decided, and so on.

But this dilemma can be resolved in the following recursive fashion: First, identify the location of T_f^{m-1} by using the values of $\hat{\alpha}_i^m$ calculated in the single-job scheme. Since only the values of the system parameters and the size of mth job are required to calculate $\hat{\alpha}_i^m$, one can easily compute $\hat{\alpha}_i^m$ by the closed-form solution described in section 2.1. Next the location of T_f^{m-1} can be decided. Once the location of T_f^{m-1} is known, *tentative* α_i^m can be computed by the algorithm described in the previous section.

In the next step, one investigates the location of T_f^{m-1} as in Fig. 6. If the location of T_f^{m-1} is changed from $\hat{\alpha}_n^m$ to α_{n+1}^m or even to $\alpha_{n+2}^m, \alpha_{n+3}^m, \ldots$, then one simply increases the value of n to the corresponding new value, $n+1, n+2, \ldots$, and calculates the values of the α_i^m again. The process is repeated until the location of T_f^{m-1} is not changed when comparing the location of T_f^{m-1} with the previous values of α_i^m and with the new values of α_i^m at the time of each iteration.

Note that when it is said that the position or location of T_f^{m-1} is changed it means that the fraction in which T_f^{m-1} is located is changed. Actually, T_f^{m-1} itself is not changed and depends only on the previous job's load.

Let us give an example. Suppose that there is a bus network with 5 processors. For

the second job, first calculate $\hat{\alpha}_i^2$ by the closed-form solution with the single-job scheme $(\hat{\alpha}_1^2, \hat{\alpha}_2^2, \ldots, \hat{\alpha}_5^2)$. If T_f^1 is located in the transmission of the second fraction $(\hat{\alpha}_2^2)$, set $n = 2$ and calculate the new values of α_i^2 by the algorithm of the multi-job scheme $(\alpha_1^2, \alpha_2^2, \ldots, \alpha_5^2)$. Now if T_f^1 is located in the transmission of the fourth fraction (α_4^2) when reidentifying the location, reset $n = 4$ and recalculate α_i^2. If the location of T_f^1 is still in the transmission of α_4^2 after recalculation and reidentification, then stop. In practice, however, the case such that the location of T_f^1 is changed after the second iteration is rare.

Note that the value of n never decreases in each iteration. It always increases. This is because the earlier parts of the fractions (e.g., α_1^2, α_2^2) become smaller and the latter parts of the fractions (e.g., $\alpha_{N-1}^2, \alpha_N^2$) become larger with each iteration.

2.5. *Is This Multi-Job Scheme Optimal?*

For a single job, the load distribution strategy is optimal if the finish time of each processor is the same. But what of the multi-job scheme? Is this optimality still valid in multi-job scheme? The answer is that for the given load distribution sequence and network configuration, the multi-job scheme is optimal.

For each individual job in the multi-job scheme, the minimum finish time occurs when all the processors finish computing at the same time, otherwise one can improve the finish time by transferring the load from busy processors to idle processors. Then how about when one considers the performance criteria to be the overall processing time of all jobs? For instance if there are 10 jobs in the system, is this multi-job scheme also optimum in terms of the 10th job's processing finish time? To answer this question in the negative one might come up with a strategy to close the gaps Γ_n to Γ_N and achieve a time performance which is less than the multi-job scheme. However there will be serious disadvantages in closing the gaps so that the result will actually be worse for the following three reasons. First, the

processing finish time will be higher for the first job and possibly some of the next few jobs. This is because except for the last job the other jobs will not have the property that each of its load fractions are finished processing at the same finish time. The second problem is that the distribution of a given job will depend not only on its own size but also on the size of the other jobs. Since this is purely deterministic system it is not causally possible to determine the size of the future job which might not be even exist. The last reason is that one cannot actually close the gaps if the computational load of the $(m-1)$st job is small (T_{cp}^{m-1} is small) and the communication load of the mth job is relatively large (T_{cm}^{m} is large). Therefore this proposed multi-job scheme is an optimal load distribution strategy.

3. ARCHITECTURE 2:

LOAD ORIGINATION AT A COMPUTING PROCESSOR

The bus network to be examined in this section is the one with load origination at a computing processor as in Fig. 7. As with the case of the network with load origination at a control processor, arriving jobs first enter a queuing system and wait for service. One of the waiting jobs in the queuing system is delivered directly to one of the N processors. Without loss of generality, we will assume this is processor 1. Processor 1 distributes the processing load of the received job among the other processors for parallel processing. Each processor is equipped with a front-end processor for communications off-loading. That is, the processors can communicate and compute at the same time. It is also assumed here that each processor may have a different computing speed.

As one might guess, the solution procedure to find the best fractions for optimal load allocation is very similar to the case of a network with load origination at a control processor. In particular, the closed-form solutions to find $\hat{\alpha}_i^m$ (single-job scheme) and α_i^1 are exactly

the same. On the other hand, the algorithm to find α_i^m for $m \geq 2$ in the multi-job scheme is similar but not identical to that for a network with load origination at a control processor. The difference is that the originating processor (processor 1) in this case does not need to transmit its own fraction (α_1^m) of the load. The following will describe the procedure to find α_i^m.

When $T_f^{m-1} \geq (1-\alpha_1^{m-1})ZT_{cm}^{m-1} + (1-\alpha_1^m)ZT_{cm}^m$ as in Fig. 8, the resulting equations are the same as in section 2.2, specifically Eq.(8) and Eq.(9). But there is a difference. In the case where the load is distributed at a control processor, the total amount of the transmission time for one job was independent of how the control processor assigns the fractions (α_i^m) or where T_f^{m-1} is located. That is, the total amount of the transmission time for the mth job was a fixed value (ZT_{cm}^m). But in the case where the load is distributed at a computing processor, it is not. Since the total amount of the transmission time where the load is distributed at a computing processor is $(\alpha_2^m + \alpha_3^m + \cdots + \alpha_N^m)ZT_{cm}^m = (1-\alpha_1^m)ZT_{cm}^m$, there is a strong dependency on the value of α_1^m. In other words, the smaller the value of α_1^m, the larger the total amount of the transmission time, and vice versa. Therefore, it is possible for the following case to exist. When first determining the location of T_f^{m-1} in the algorithm in section 2.4 by using the values of the load allocation fractions in the single-job scheme $(\hat{\alpha}_2^m, \hat{\alpha}_3^m, \ldots, \hat{\alpha}_N^m)$, T_f^{m-1} is located after the time that processor 1 finishes the transmission of the mth job's load. So one might calculate the new values of the fractions (α_i^m) simply by using Eq.(8) and Eq.(9). But when identifying the location of T_f^{m-1} again after calculation, it is possible that the new location of T_f^{m-1} is placed before the end of the transmission as in Fig. 9. This is because, as mentioned before, the early fractions becomes smaller and the latter fractions becomes larger in the multi-job scheme compared to the single-job scheme. If this happens, the situation becomes the case that $T_f^{m-1} < (1-\alpha_1^{m-1})ZT_{cm}^{m-1} + (1-\alpha_1^m)ZT_{cm}^m$.

One more thing that must be pointed out is that as one can see in Fig. 9 it takes less time to transmit the mth job's load in the single-job scheme than in the multi-job scheme because

the multi-job's total transmission time is longer than the single-job's. However, the total processing time $(T_f^m - T_f^{m-1})$ to compute the mth job's load in the single-job scheme takes longer than that in the multi-job scheme because if the load allocation fractions calculated in the single-job scheme are used in the multi-job scheme then, for jobs after the first job, each processor does not finish its computation at the same time. This increases the solution time. This is because the single job load fractions are based on the assumption that there is waiting time.

When $T_f^{m-1} < (1 - \alpha_1^{m-1})ZT_{cm}^{m-1} + (1 - \alpha_1^m)ZT_{cm}^m$, the resulting equations are very similar to those in section 2.3, specifically Eq.(18) to Eq.(21). The only difference is that the summation term in the numerator of u_i^m is the sum of $\frac{w_1}{w_k}$ from $k = 2$ to $n - 1$ or $\sum_{k=2}^{n-1} \frac{w_1}{w_k}$ in the case where the load is distributed at a control processor while the summation term in the numerator of u_i^m is the sum of $\frac{w_1}{w_k}$ from $k = 1$ to $n - 1$ or $\sum_{k=1}^{n-1} \frac{w_1}{w_k}$ in the case where the load is distributed at a computing processor.

4. PERFORMANCE EVALUATION

Based on the previous results, some performance evaluation results were computed for various cases – the single-job scheme and the multi-job scheme for the network with load origination at a control processor and the network with load origination at a computing processor. In each plot, the total solution finish time is drawn against the number of jobs. In this section, the total number of presented jobs is 10 and no new jobs arrive. That is, the queue initially contains 10 jobs. There are five processors in the network, and it is assumed that the values of the channel speed, the computing speed of each processor and communication load for every job are assigned to unity (Z, all w_i's, $T_{cm}^m = 1$ for all m).

- In Fig. 10, the total solution finish time is plotted against the number of jobs when

the computational load of every job is 6 ($T_{cp}^m = 6$ for all m) which is a relatively medium load in comparison with communication load which is set to one and with five processors. Since the computational load for every job is the same, the total solution finish time is linear in the single-job scheme. Clearly, the the multi-job scheme is superior to the single-job scheme in terms of the total solution finish time.

- Fig. 11 and Fig. 12 plot the solution time when the computational load of the first half of 10 jobs is 10 ($T_{cp}^m = 10$ for $m = 1, 2, \cdots, 5$, heavy computational load) and that of the second half of 10 jobs is 3 ($T_{cp}^m = 3$ for $m = 6, 7, \cdots, 10$, light computational load), and vice versa for the single-job scheme and for the multi-job scheme for the network with load origination at a control processor and for the network with load origination at a computing processor. Naturally, the total solution finish time of the last job in the single-job scheme is the same no matter what order the computational load is assigned although there is some difference in the total solution finish time in the middle of the processing of the given jobs. However, for the multi-job scheme assigning the job with the heavy computational load first and the one with light computational load last causes a reduction of the total solution finish time at the end of processing. This is because serving the heavy computational load first results in a timing diagram like Fig. 4 rather than Fig. 5. If there are several heavy jobs ahead of the light jobs, the load distribution will be done much ahead of the time of its computation and this means there is no wasted time or less waiting time between the consecutive jobs. Therefore, if it is possible to assign the order of service at one's convenience, i.e., the service discipline is not restricted to FIFO, assigning heavy computational load first can reduce the overall solution finish time.

5. CONCLUSION

In this paper, an efficient strategy for optimal load allocation for bus networks with multiple jobs is discussed. Starting from the second job in the multi-job scheme, it is found that one can reduce the total solution finish time by comparison with the single-job scheme. It is also found that in the multi-job scheme the overall job computing finish time can be futher reduced by assigning jobs with a heavy computational load job first.

Acknowledgement

The research in this paper was supported in part by the BMDO/IST under the U.S. Office of Naval Research under grant no. N00014-91-J4063.

References

[1] Bataineh, S., and Robertazzi, T. G. Distributed computation for a bus network with communication delays. *Proc. 1991 Conference on Information Sciences and Systems.* The Johns Hopkins University, Baltimore, MD, 1991, pp. 709–714.

[2] Bataineh, S., and Robertazzi, T. G. Bus oriented load sharing for a network of sensor driven processors. *IEEE Trans. Systems, Man and Cybernetics.* **21**, (1991), 1202–1205.

[3] Bataineh, S., and Robertazzi, T. G. Ultimate performance limits for networks of load sharing processors. *Proc. 1992 Conference on Information Science and Systems.* Princeton University, Princeton, NJ, 1992, pp. 794–799.

[4] Bataineh, S., Hsiung, T., and Robertazzi, T. G. Closed form solutions for bus and tree networks of processors load sharing a divisible job. *International Conference on Parallel Processing.* St. Charles, IL, 1993, Accepted by the IEEE Trans. Computers.

[5] Bharadwaj, V., Ghose, D., and Mani, V. Optimal sequencing and arrangement in distributed single-level tree networks with communication delays. accepted by the IEEE Trans. Parallel and Distributed Systems.

[6] Bharadwaj, V., Ghose, D., and Mani, V. An efficient load distribution strategy for a distributed linear network of processors with communication delays. accepted by the Computer and Mathematics with Applications.

[7] Bharadwaj, V., Ghose, D., and Mani, V. Installment techniques in tree networks. accepted by the IEEE Trans. Aerospace and Electronic System.

[8] Blazewicz, J., Drabowski, M., and Weglarz, J. Scheduling multiprocessor tasks to minimize schedule length. *IEEE Trans. Computers.* **C-35** (1986), 389–398.

[9] Bokhari, S. H. *Assignment problems in parallel and distributed computing.* Kluwer Academic Publishers, Boston, 1987.

[10] Cheng, Y. C., and Robertazzi, T. G. Distributed computation with communication delays. *IEEE Trans. Aerospace and Electronic Systems.* **24**, (1988), 700–712.

[11] Cheng, Y. C., and Robertazzi, T. G. Distributed computation for a tree network with communication delays. *IEEE Trans. Aerospace and Electronic Systems.* **26**, (1990), 511–516.

[12] Chong, C. Y., Tse, E., and Mori, S. Distributed estimation in networks. *presented at the American Control Conference.* (San Francisco).

[13] Du, J., and Leung, J. Y. T. Complexity of scheduling parallel task systems. *SIAM Journal on Discrete Mathematics.* (1989), 473–487.

[14] Ghose, D., and Mani, V. Distributed computation in a linear network: Closed-form solutions and computational techniques. *IEEE Trans. Aerospace and Electronic Systems.* **30**, (1994).

[15] Ghose, D., and Mani, V. Distributed computation with communication delays: Asymptotic performance analysis. accepted by the J. Parallel and Distrib. Comput.

[16] Kim, H. J., Jee, G. I., and Lee, J. G. Optimal load distribution for tree network processors. submitted for publication.

[17] Robertazzi, T. G. Processor equivalence for a linear daisy chain of load sharing processors. *IEEE Trans. Aerospace and Electronic Systems.* **29**, (1993), 1216–1221.

[18] Sohn, J. and Robertazzi, T. G. Optimal load sharing for a divisible job on a bus network. *Proc. 1993 Conference on Information Science and Systems.* The Johns Hopkins University, Baltimore, MD, 1993.

[19] Stone, H. S. Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Trans. Software Engineering.* **SE-3**, (1977), 85–93.

[20] Tenney, R. R., and Sandell, N. R., Jr. Detection with distributed sensors. *IEEE Trans. Aerospace and Electronic Systems.* **AES-17**, (1981), 501–510.

[21] Zhao, W., Ramamritham, K., and Stankovic, J. A. Preemptive scheduling under time and resource constraints. *IEEE Trans. Computers.* **C-36**, (1987), 949–960.

Figure Captions

Figure 1. Bus network with load origination at a control processor.

Figure 2. Timing diagram for the bus network with load origination at a control processor in the single-job scheme.

Figure 3. Timing diagram for the bus network with load origination at a control processor in the multi-job scheme.

Figure 4. Timing diagram for the bus network with load origination at a control processor in the multi-job scheme when $T_f^1 \geq ZT_{cm}^1 + ZT_{cm}^2$.

Figure 5. Timing diagram for the bus network with load origination at a control processor in the multi-job scheme when $T_f^1 < ZT_{cm}^1 + ZT_{cm}^2$.

Figure 6. Transmission timing diagram for the bus network with load origination at a control processor in the single-job scheme and in the multi-job scheme.

Figure 7. Bus network with load origination at a computing processor.

Figure 8. Timing diagram for the bus network with load origination at a computing processor in the multi-job scheme when $T_f^1 \geq (1-\alpha_1^1)ZT_{cm}^1 + (1-\alpha_1^2)ZT_{cm}^2$.

Figure 9. Transmission timing diagram for the bus network with load origination at a computing processor in the single-job scheme and in the multi-job scheme.

Figure 10. Total solution time when $T_{cp}^m = 6$ for all m.

Figure 11. Total solution time for network with load origination at a control processor when $T_{cp}^m = 10$ for $m = 1, 2, \ldots, 5$ and $T_{cp}^m = 3$ for $m = 6, 7, \ldots, 10$, and vice versa.

Figure 12. Total solution time for network with load origination at a computing processor when $T_{cp}^m = 10$ for $m = 1, 2, \ldots, 5$ and $T_{cp}^m = 3$ for $m = 6, 7, \ldots, 10$, and vice versa.

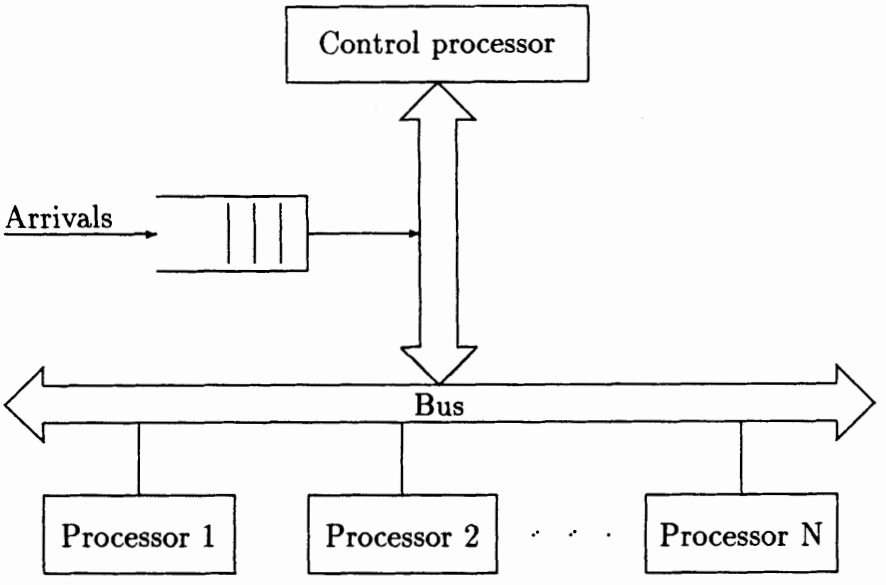

Figure 1. Bus network with load origination at a control processor.

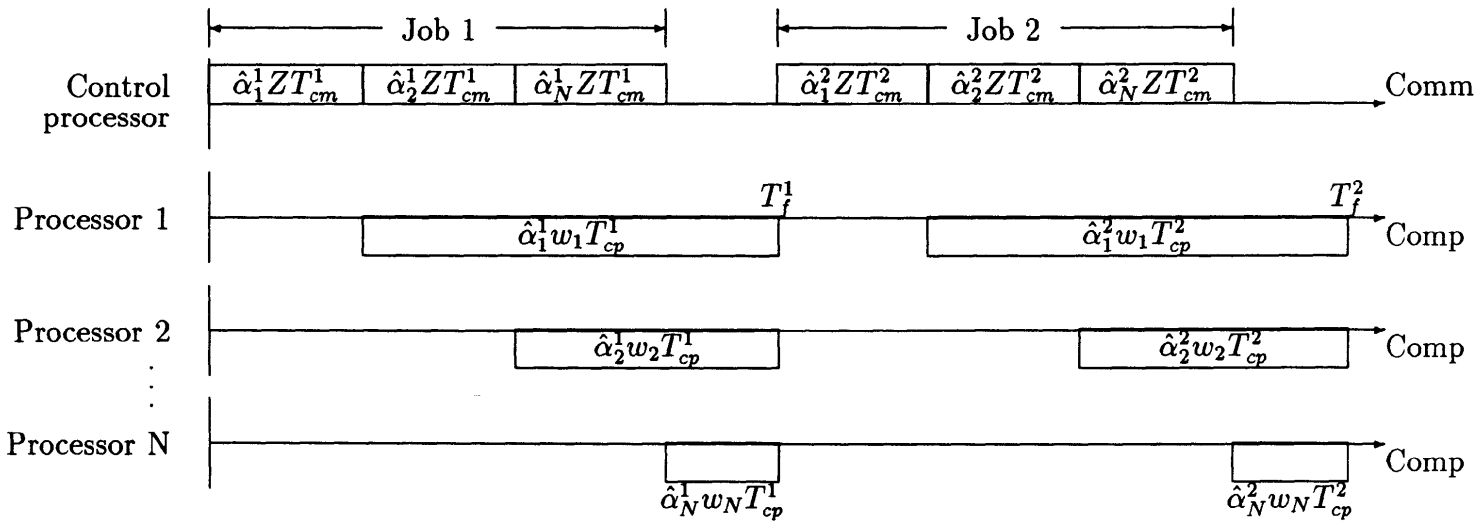

Figure 2. Timing diagram for the bus network with load origination at a control processor in the single-job scheme.

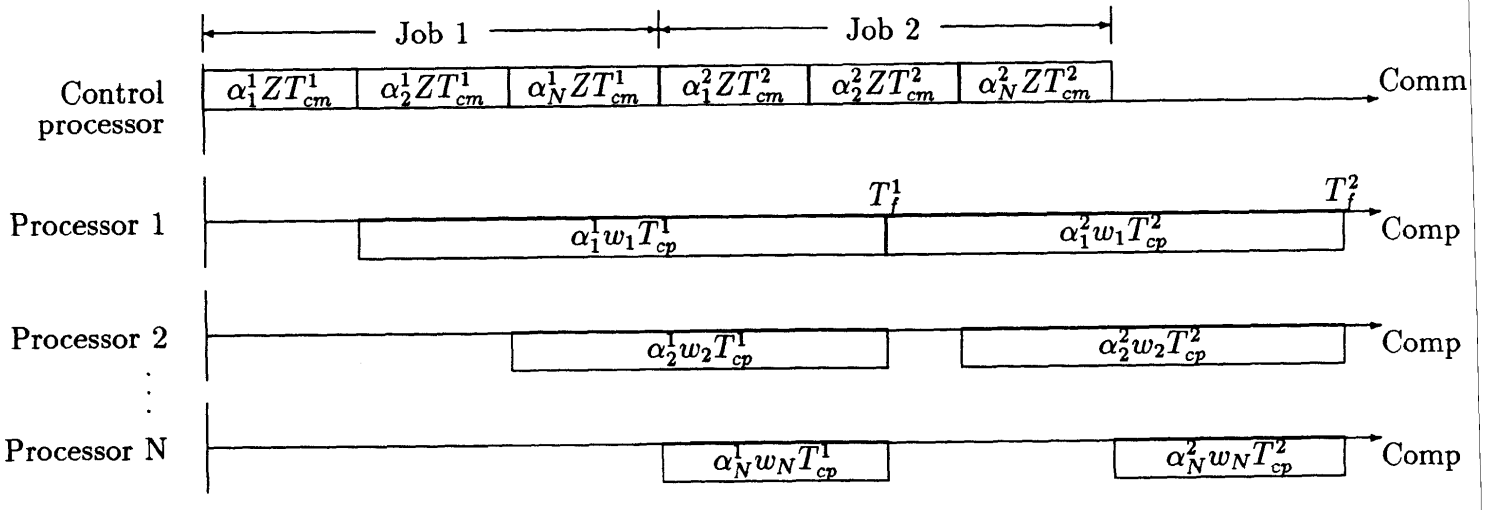

Figure 3. Timing diagram for the bus network with load origination at a control processor in the multi-job scheme.

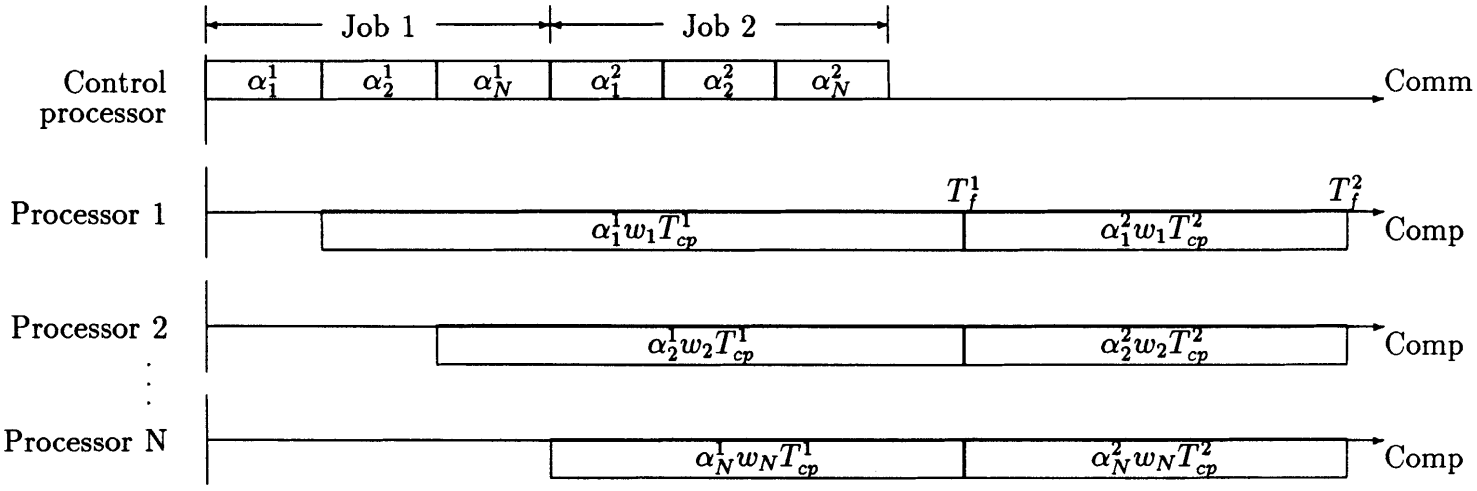

Figure 4. Timing diagram for the bus network with load origination at a control processor in the multi-job scheme when $T_f^1 \geq ZT_{cm}^1 + ZT_{cm}^2$.

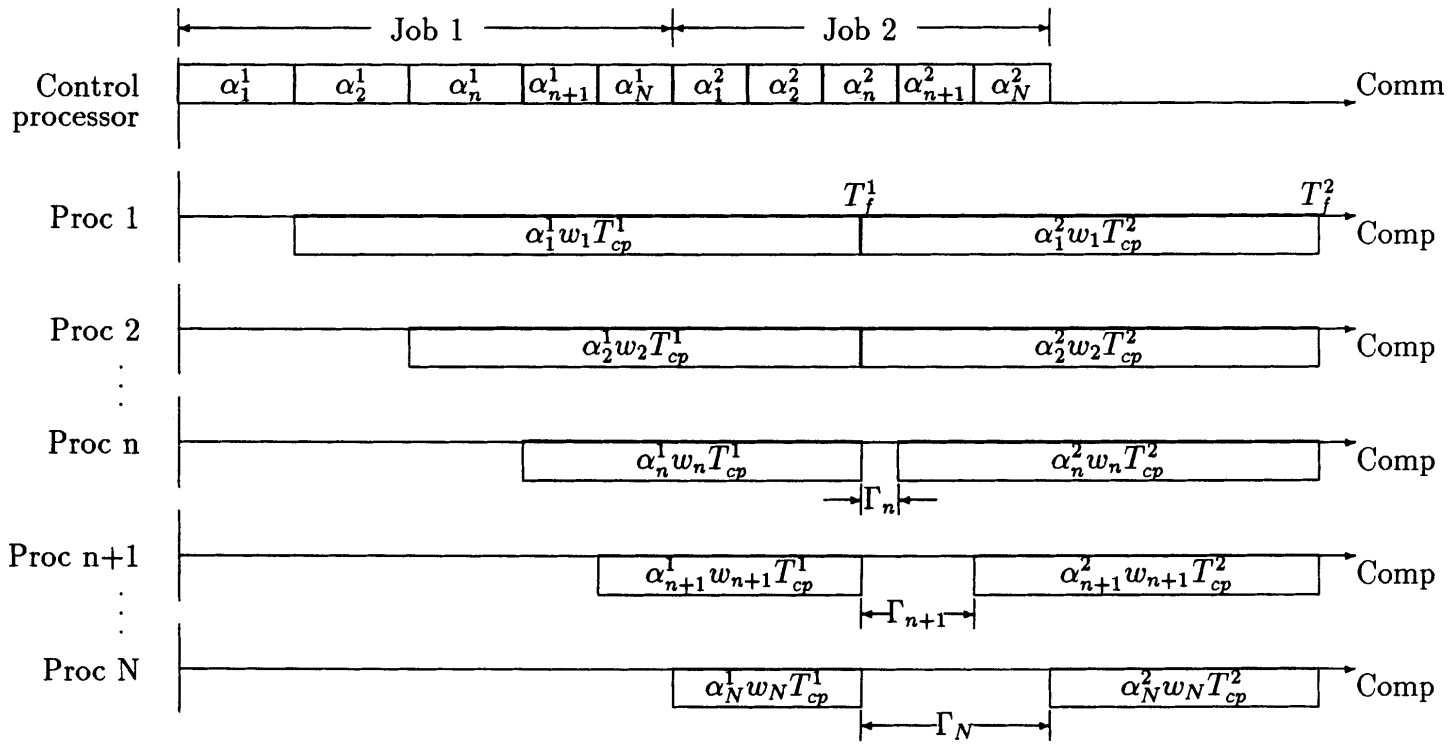

Figure 5. Timing diagram for the bus network with load origination at a control processor in the multi-job scheme when $T_f^1 < ZT_{cm}^1 + ZT_{cm}^2$.

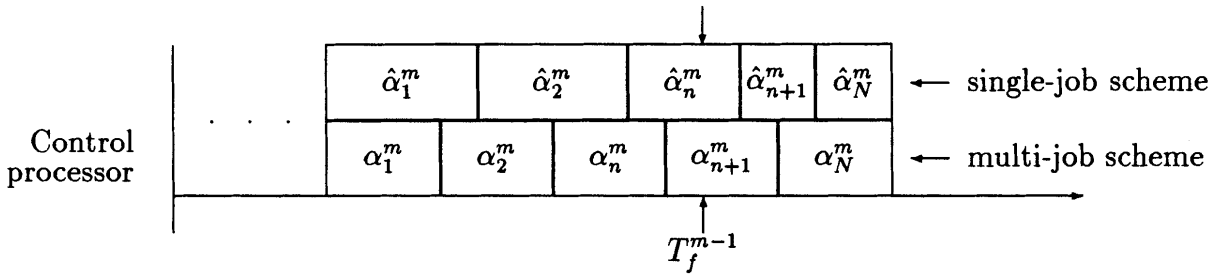

Figure 6. Transmission timing diagram for the bus network with load origination at a control processor in the single-job scheme and in the multi-job scheme.

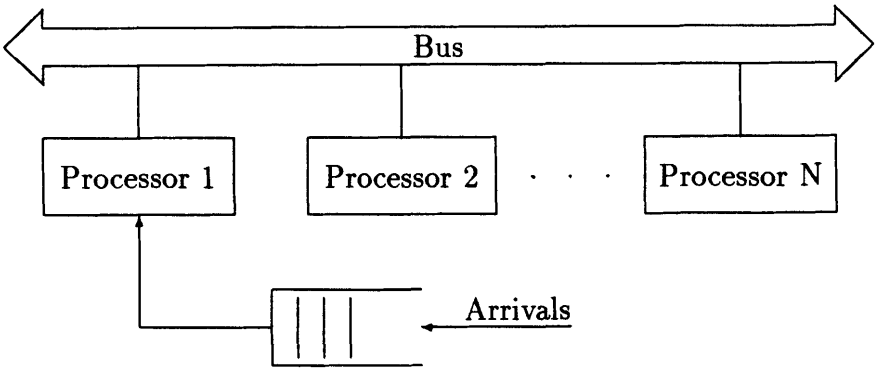

Figure 7. Bus network with load origination at a computing processor.

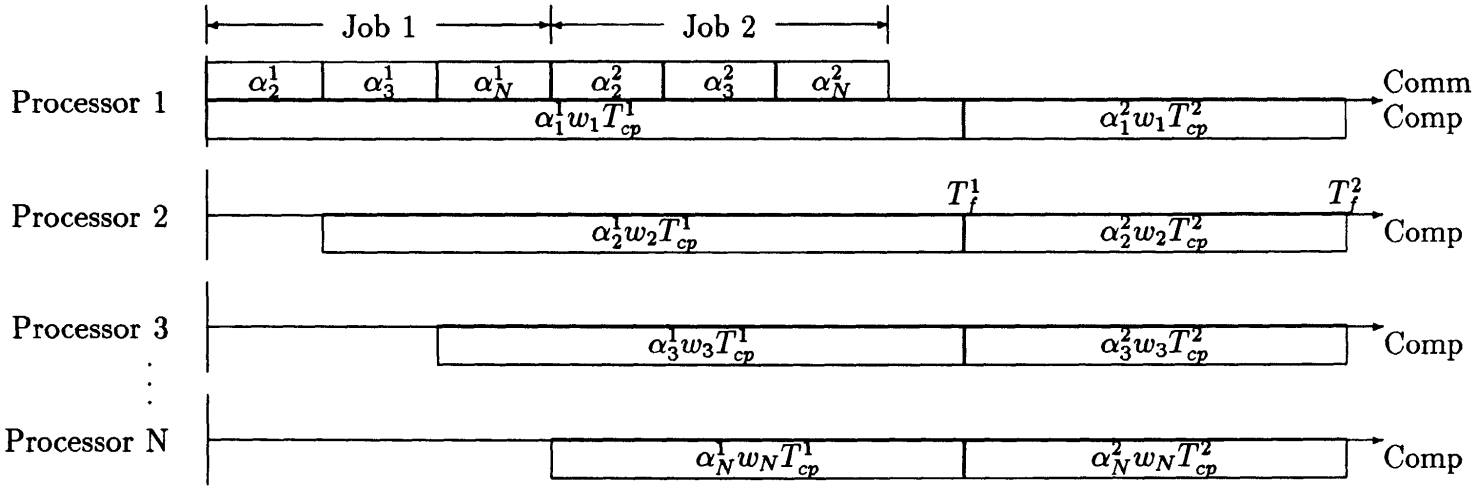

Figure 8. Timing diagram for the bus network with load origination at a computing processor in the multi-job scheme when $T_f^1 \geq (1-\alpha_1^1)ZT_{cm}^1 + (1-\alpha_1^2)ZT_{cm}^2$.

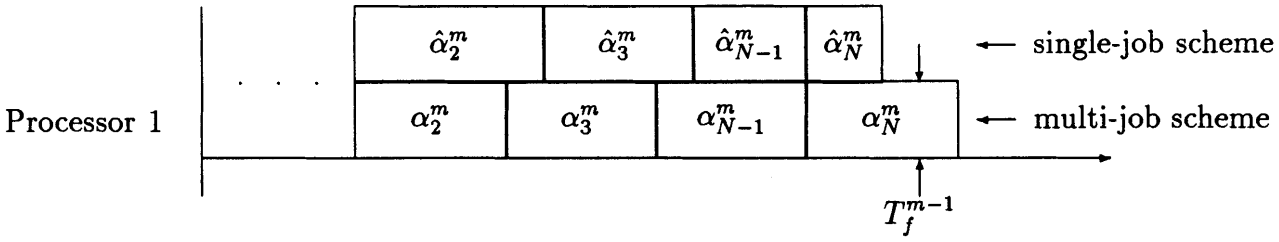

Figure 9. Transmission timing diagram for the bus network with load origination at a computing processor in the single-job scheme and in the multi-job scheme.

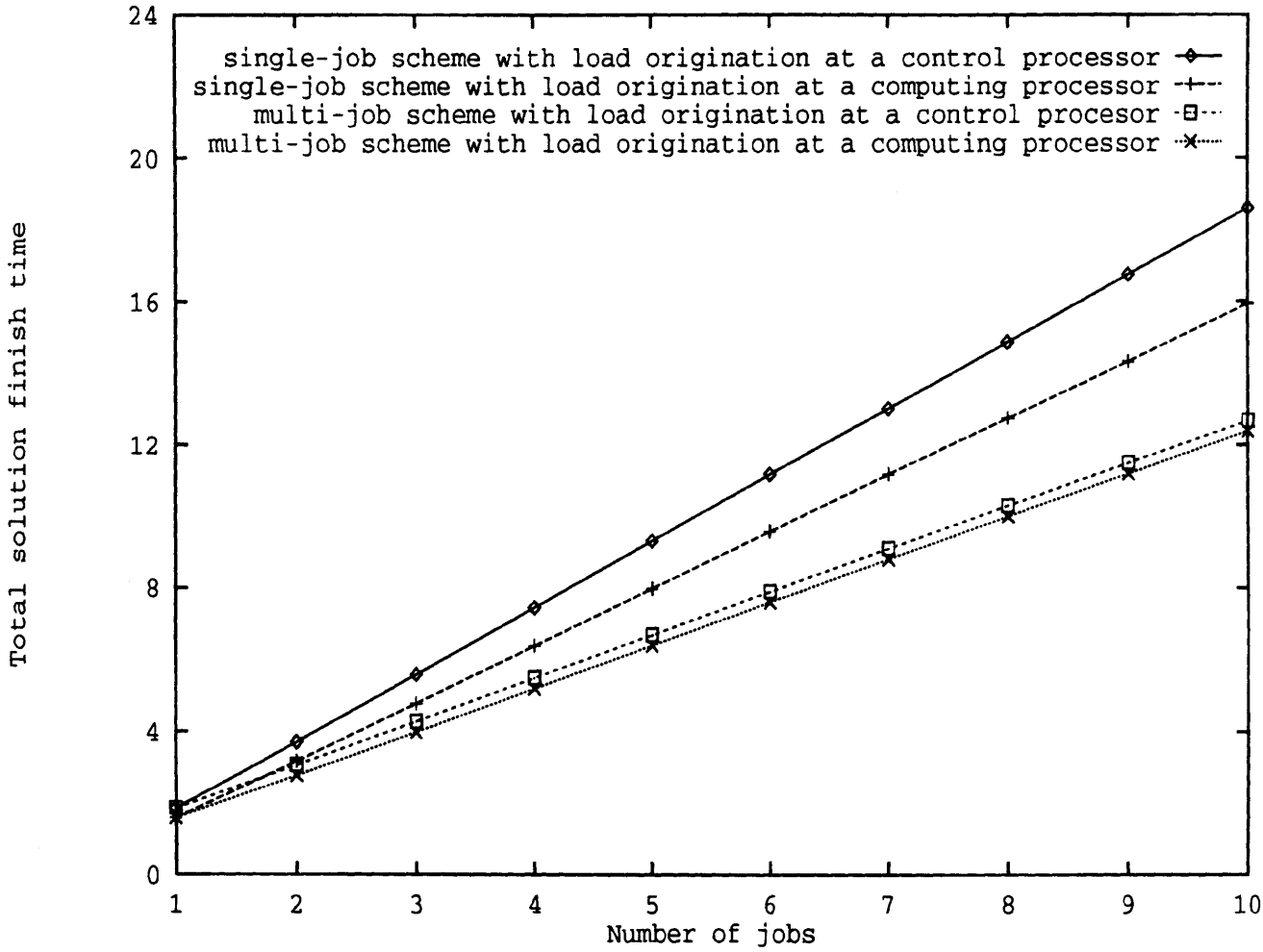

Figure 10. Total solution time when $T_{cp}^{m} = 6$ for all m.

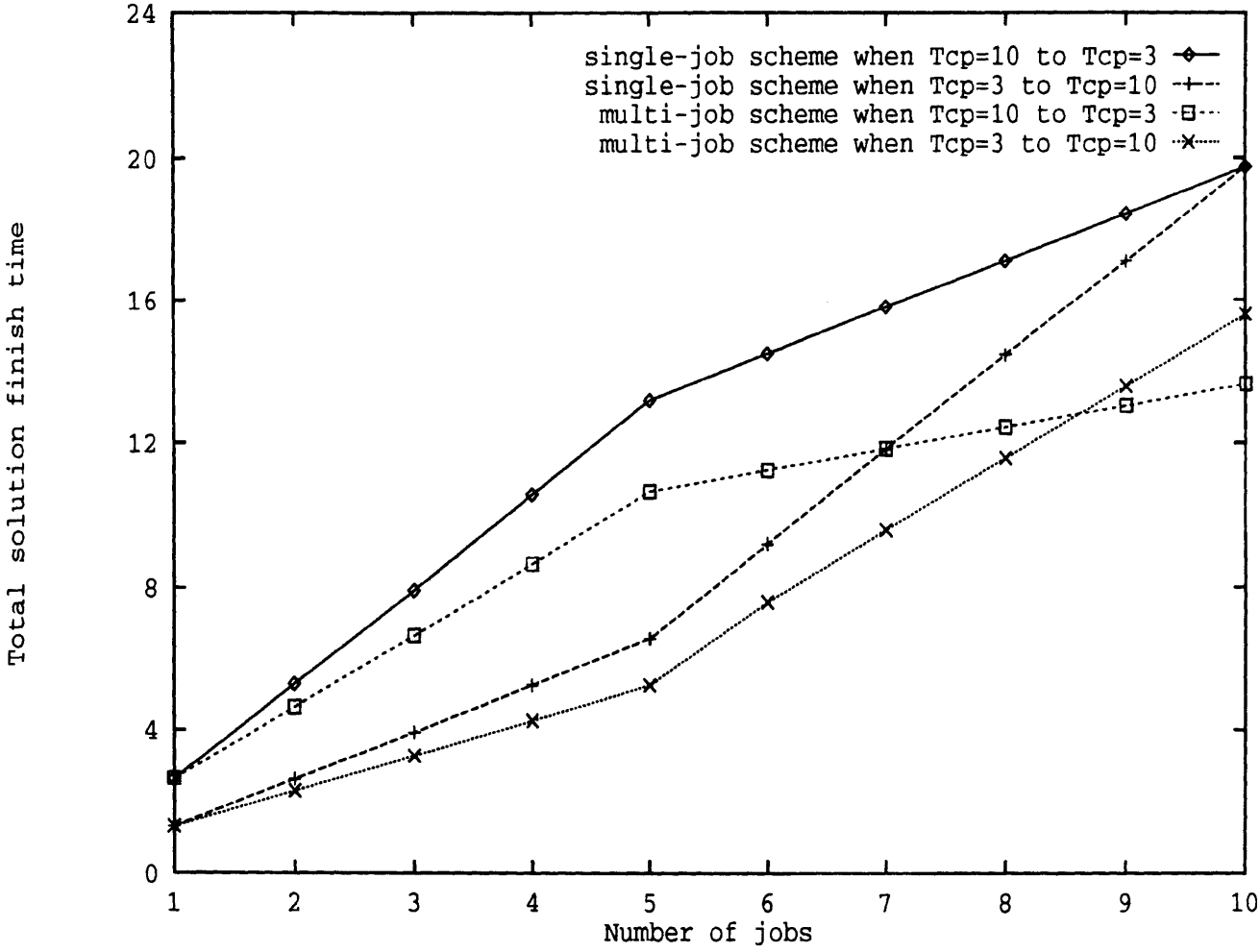

Figure 11. Total solution time for network with load origination at a control processor when $T_{cp}^m = 10$ for $m = 1, 2, \ldots, 5$ and $T_{cp}^m = 3$ for $m = 6, 7, \ldots, 10$, and vice versa.

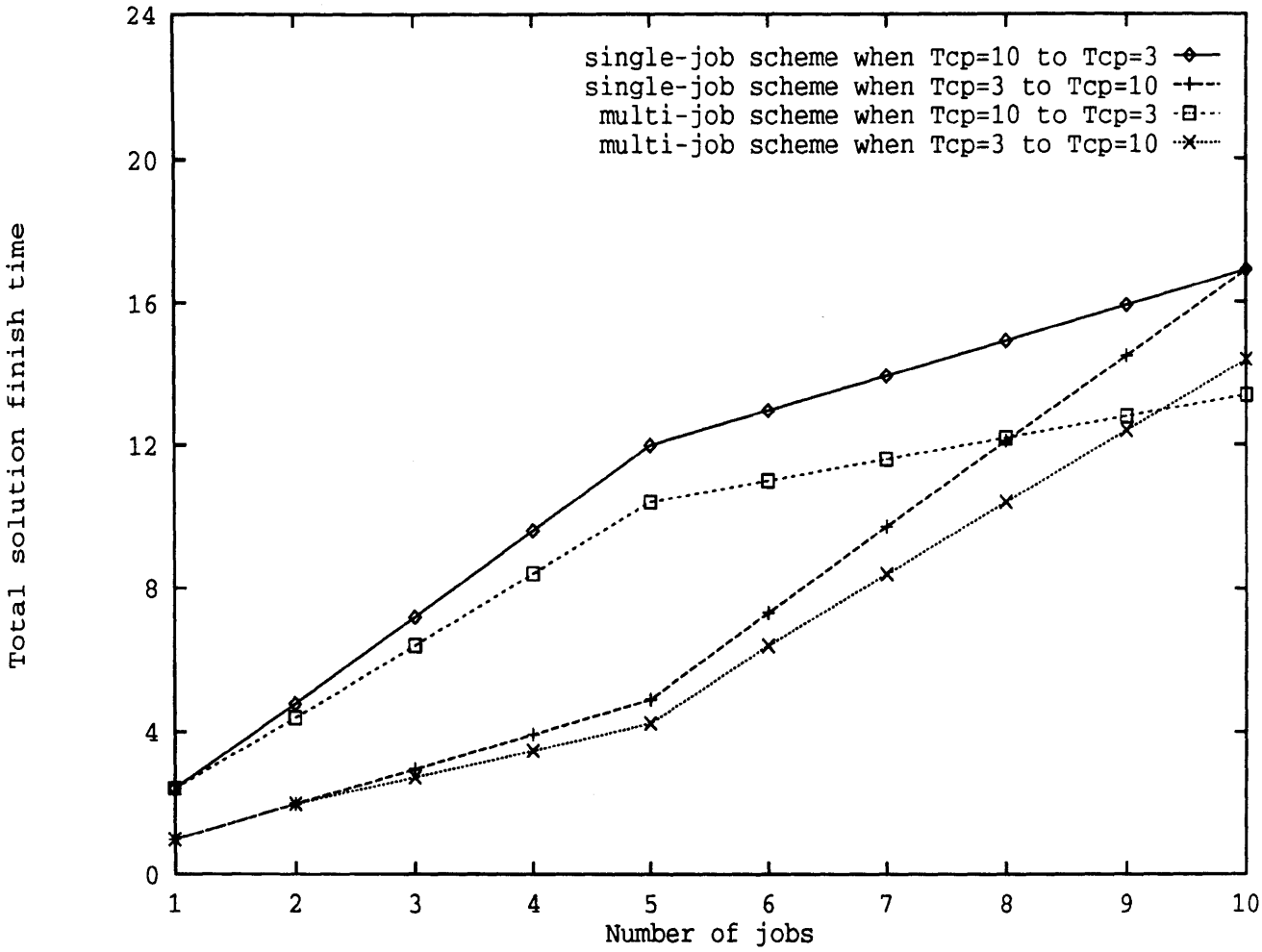

Figure 12. Total solution time for network with load origination at a computing processor when $T_{cp}^{m} = 10$ for $m = 1, 2, \ldots, 5$ and $T_{cp}^{m} = 3$ for $m = 6, 7, \ldots, 10$, and vice versa.