Suboptimal Solutions Using Integer Approximation Techniques for Scheduling Divisible Loads on Distributed Bus Networks

Bharadwaj Veeravalli, *Member, IEEE,* and N. Viswanadham, *Fellow, IEEE*

***Abstract*—The problem of optimal divisible load distribution in distributed bus networks employing a heterogeneous cluster of processors is addressed. The objective is to minimize the total processing time of the entire load subject to the communication and computation delays. In the mathematical model we adopt, both the granularity of the load fractions and all the associated overheads (also referred to as start-up costs) in the process of communication and computation, are considered explicitly in the problem formulation. We introduce a *directed flow graph* model for representing the load distribution process. This representation is novel to this literature. With this model, we first derive a closed-form solution for an optimal processing time. We propose an integer approximation algorithm and derive ultimate performance bounds for the class of homogeneous networks. We then extend the problem to a special class of application problems in which the data partitioning is restricted to a finite number of partitions. For this case, we present a recursive procedure to obtain optimal processing time. We then present two different integer approximation algorithms—PIA and IIA that could generate integer load fractions and yield suboptimal solutions. The choice of these algorithms are also analyzed. All the results are extended to a class of homogeneous networks to obtain ultimate performance bounds. Several illustrative examples are provided for ease of explanation.**

***Index Terms*—Communication delays, directed graph, divisible loads, granularity, integer approximation, processing time.**

I. Introduction

RESEARCH in the area of *divisible load theory* (DLT) has demonstrated a significant impact both in terms of the contributions to the domain of parallel and distributed scheduling literature, as well as in terms of the applicability of the results to a wide range of research disciplines [2]–[8], [10], [11]. This multidisciplinary research has attracted a large pool of researchers since 1988 through the first introductory work on bus based multiprocessor systems by Cheng and Robertazzi [2]. The main attraction is due to its simplicity in formulating and modeling the problem in a linear fashion, at the same time producing a richer set of results that elicits many interesting properties for network based scheduling schemes. The mathematical model for the communication and computation delays are assumed to be proportional to the size of the load that the link carries and the processor computes. The processing load in this domain is assumed to be arbitrarily divisible with no precedence relationships between the partitioned fractions. Also, they are computationally intensive and hence, partitioning and scheduling the entire load among the available set of processors on the network minimizes the overall finish time. However, owing to the communication and computation delay constraints on the links and the processors, the partitioning schemes that are usually practiced in the parallel processing literature do not produce optimal, if not acceptable performance, in scheduling these divisible loads. The performance metric [13] considered in these minimization problems is the total processing time of the entire load. The primary objective of this theory is to decide on how to partition the available load and schedule among the processors so that the processing time is minimized.

We will now present a very brief summary on some of the relevant recent literature in DLT. In the literature so far, the above mentioned problem has been studied in terms of architectural variations such as , single-level tree networks [3], [7], linear networks [4], [14], [17], mesh [5], [6], hypercubes [6], bus [2], [7], [8], etc. A compilation of the results until 1995 appears in [7]. Also, issues related to multiprocessor systems such as fault-tolerance [16], scheduling loads subject to the availability of processors, referred to as release times [19], etc are also studied. The time varying nature of the processor and channel speeds is considered by Jeeho *et al.* [19] in the design and analysis of load distribution strategies. In [20], [21], using additive overhead factors that influence the communication and computation times, a closed-form solution for an optimal processing time is derived and the effect of load sequencing is rigorously carried out. Also, in [20], [21], an integer approximation technique is proposed and ultimate time performance bound is derived. We have presented that algorithm in Section III in this paper for the purpose of continuity. However, very recently the trend in this domain has been geared toward tackling some realistic applications that are often found in practice. Very recently the problem of matrix–vector product computations for very large size matrices are analyzed using the strategies developed in DLT by Ghose *et al.* [8]. Alternate to this processing time minimization problem, Jeeho *et al.* [15] considered the minimization problem from monetary cost perspective. This problem considers the minimization of the total monetary cost incurred in scheduling divisible loads on bus networks.

Manuscript received June 20, 1999; revised September 23, 2000. This work was supported in part by the Department of Electrical and Computer Engineering, The National University of Singapore, under research Grant R-263-000-073-112. This paper was recommended by Associate Editor K. Pattipati.

V. Veeravalli is with the Department of Electrical and Computer Engineering, The National University of Singapore, Singapore 119260 (e-mail: elebv@nus.edu.sg).

N. Viswanadham is with the Department of Mechanical and Production Engineering, The National University of Singapore, Singapore 119260 (e-mail: mpenv@nus.edu.sg).

Publisher Item Identifier S 1083-4427(00)10875-6.

A. Research Contributions

Our research contributions in this paper are multifold and are novel to this literature in DLT. This research is an attempt to bring the results of the theory so far closer to practice by tuning the model and in proposing a load distribution strategy and solution that can be realized in practice. This contribution precisely gives an estimate of how far the solution proposed by the theory will lie in comparison with the realistic situation. To this end, we tackle two different, but closely related problems. We first introduce a *directed flow* graph (DFG) representation to describe the load distribution process. We believe that this directed flow graph is a generalized representation. This is the first time in this domain that such a representation is introduced. This representation provides complete flexibility in representing any complex schedule and also to analyze its performance. Apart from serving as an alternative tool to the timing diagram representation [7], this representation is simple to use and deriving an optimal solution is less cumbersome. We use this representation throughout this paper. We derive a closed-form solution for an optimal processing time using a mathematical model that accounts for all the overhead components that penalize the performance. We show the impact of these overheads that are present in reality in scheduling divisible loads. Though this model was considered earlier in the literature [6], closed-form solutions, load sharing conditions, and the impact of these overheads on the performance were not analyzed. Secondly, for a class of applications that demand the load to be divided into a finite number of partitions, we first derive real valued optimal load distribution and the processing time.

Thirdly, from an applications perspective, in general, the processing load may not be truly arbitrarily divisible. An example of this scenario would be in computing a large size matrix–vector product on parallel and distributed systems [8] or in any image processing application [13]. A large size matrix or a vector may be partitioned for the ease of complex computations. In this case, the arbitrarily divisible property no longer holds as the load that will be distributed among the processors will be in terms of number of rows or columns and not as a fraction of the rows or columns. Thus, if our model can be tuned to fit this requirement, then the strategies proposed by this theory would suit any application. For instance, in the above mentioned applications, usually the load that is assigned to a processor will be in terms of certain number of rows or columns and hence, we can say that the load that is assigned to a processor will be an integral multiple of some fundamental quantity. This fundamental quantity is referred to as the *divisibility factor* and is defined as the minimum possible *granularity* of any load fraction that can be assigned to a processor. In practice, most of the times the choice of an appropriate grain size (granularity) during the runtime becomes an important factor, as the effective speed of the network and the processors may vary due to several effects. Thus, one of the main concerns of this paper is to propose some integer approximation techniques to generate integer valued sizes of the load. One may employ a simple rounding off procedure, however, in this case, there is no guarantee that the performance will be within acceptable limits and also it is difficult to quantify by means of some performance bounds. Our proposed algorithms follow a systematic way of generating the integer valued load fractions and guarantee that the time performance converges within an acceptable radius from the optimal solution for the case of homogeneous bus networks. This is the first time that the effect of integer approximation techniques are studied in the DLT literature.

The paper is organized as follows. Section II introduces the problem and describes the load distribution strategy. It also introduces the DFG representation for scheduling and derives a closed-form solution for the processing time. Section III proposes an integer approximation algorithm and obtains an ultimate time performance bound. Section IV tackles another closely related problem in which partitioning the load is restricted to a finite number. Also, in this section, we propose two different integer approximation algorithms and obtain ultimate performance bounds. Section V discusses the results obtained and Section VI concludes the paper.

II. Problem Setting and Some Remarks

We consider a bus network architecture as shown in Fig. 1. The network may or may not have a dedicated control processor or a bus controller unit (BCU) [7] to distribute the load among the processors. In this paper, we shall present a rigorous analysis of the time performance of the system for the case when the network has a dedicated BCU to distribute the entire load. It is worth mentioning at this stage that a bus network is equivalent to a single level tree network or a star network when all the links have identical speeds [3], [7]. Thus, all the research contributions in this paper also hold for this special class of single-level tree networks.

A. Load Distribution Strategy and Some Definitions

The load distribution strategy is described as follows. The divisible load is assumed to originate at the BCU. The BCU divides the load into m load fractions, denoted as $\alpha_1, \cdots, \alpha_m$, and distributes them among all the m-processors in a particular sequence, say $p_1, \cdots, p_m$, one after other. Upon receiving their respective load fractions, the processors start computing their respective load fractions. The problem is then to determine the optimal sizes of these load fractions that are assigned to the processors such that the total processing time is a minimum. This strategy is referred to as *single installment* strategy in the literature [7]. We now introduce some notations that will be used throughout the paper.

α_i	Fraction of the load assigned to processor p_i.
w_i	The inverse of the computation speed of processor p_i.
T_{cp}	Time taken to process a unit load by the standard processor.
θ_{cp}	An additive computation overhead component that includes the sum of all delays associated with the computation process.
z	The inverse of the communication speed of the link.
T_{cm}	Time taken to transmit a unit load by the communication link.
θ_{cm}	An additive communication overhead component that includes the sum of all delays associated with the communication process.

We shall denote the product $w_i T_{cp}$ as E_i and zT_{cm} as C, respectively throughout the paper. Thus, using the above notations, we

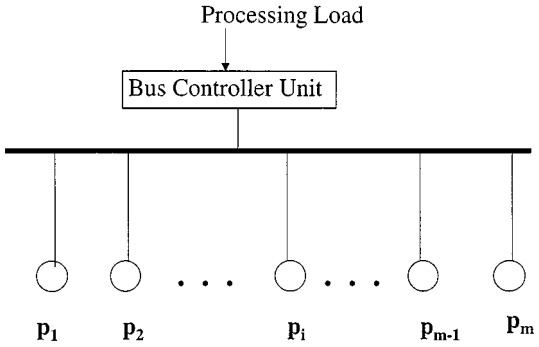


Fig. 1. Distributed bus network with m processors and a bus control unit (BCU).

see that the communication time of a fraction of the load α_i is given by $\alpha_i C + \theta_{cm}$ and the computation time of this load fraction by p_i is given by, $\alpha_i E_i + \theta_{cp}$ [2]–[19].

B. Directed Flow Graph

The load distribution process is described by means of a directed flow graph as shown in Fig. 2. The figure shows two types of nodes. The first level of nodes are referred to as *communication nodes* and the other nodes are referred to as *computation nodes*. The weight of a communication node i is given by $\alpha_i C + \theta_{cm}$ and the weight of the computation node i is given by $\alpha_i E_i + \theta_{cp}$. The directed arrows between the adjacent communication nodes i and $i+1$ represent the fact that the communication of the load fraction α_{i+1} to p_{i+1} will start only after the communication of the fraction α_i to p_i is completed. Similarly, the directed arrows between a communication node i and the computation node i denotes the fact that the computation of α_i by p_i starts only after receiving the entire load α_i from BCU. These directed arrows represent the *causal precedence* relationships between the events. Thus, we see that this directed flow graph represents the load distribution process completely. The main advantage of this representation is its simplicity. In the DLT theory, as a principle of optimality, an optimal solution is obtained when all the participating processors stop computing at the same time. This fact is also captured in this flow graph by equating the finish time paths (defined in Section II-B) and solving the recursive equations. The timing diagram representation can be elegant and easy to visualize only for simple load distribution strategies. Using a timing diagram to represent an optimal schedule for complex strategies, like multi-installment strategy, is extremely difficult and time consuming. As mentioned earlier, we use this representation to derive optimal processing time throughout the paper.

C. Some Definitions

We shall now define the following.

i) **Load distribution**, denoted by α, defined as an m-tuple $(\alpha_1, \cdots, \alpha_m)$ such that $0 < \alpha_i \leq 1$ and $\sum_{i=1}^{m} \alpha_i = L$.
The equation $\sum_{i=1}^{m} \alpha_i = L$ is referred to as *normalization equation*, where L is the total load. Let the space of all possible load distributions be denoted as Γ.

ii) **Finish time path** of a processor p_i, denoted as $P_i(\alpha, m)$, is the sum of the weights of the nodes starting from the communication node 1 till computation node i, along the directed arrows.

iii) **Critical path**, denoted as $P(\alpha, m)$, and is given by $P(\alpha, m) = \max\{P_i(\alpha, m)\}$, $i = 1, 2, \cdots, m$, where P_i is as defined above.
This is the longest finish time path in the graph and represents the time at which the entire load is processed.

iv) **Optimal path**, denoted as $P^*(\alpha^*, m)$, which is the minimum processing time to finish processing the entire load, i.e., $P^*(\alpha^*, m) = \min_{\alpha \in \Gamma}\{P(\alpha, m)\}$.

It has been rigorously proved in the literature [7] that *for optimal processing time all the finish time paths of the processors must be equal*. Here too, we shall use this optimality criterion to analyze the processing time performance of the system.

D. Closed-Form Solutions for the Optimal Processing Time

In this section, we shall derive a closed-form expression for an optimal processing time by assuming that the sequence of load distribution is from p_1 to p_m in that order. Further, we shall show that the presence of such overheads, which are inherently present in any realistic system, will drastically affect the time performance and may lead to different design decisions. This is crucial whenever the optimality of the solution and related trade-off studies are important in the design. A detailed discussion is presented in Section VI. Throughout this section, this sequence of load distribution will be referred to as a *fixed sequence*.

From Fig. 2, for an optimal solution, equating the finish time paths $P_i(\alpha, m)$ and $P_{i+1}(\alpha, m)$, we obtain the following recursive equations:

$$\alpha_i E_i + \theta_{cp} = \alpha_{i+1}(E_{i+1} + C) + \theta_{cp} + \theta_{cm}, \quad i = 1, \cdots, m-1. \tag{1}$$

We rewrite (1) as

$$\alpha_i = \alpha_{i+1} f_{i+1} + \beta_i, \qquad i = 1, \cdots, m-1, \tag{2}$$

where $((E_{i+1} + C)/E_i) = f_{i+1}$, and $(\theta_{cm}/E_i) = \beta_i$, for all $i = 1, \cdots, m-1$. Now, expressing each of these load fractions in terms of α_m, we obtain,

$$\alpha_i = \alpha_m M_i + N_i, \qquad i = 1, \cdots, m-1 \tag{3}$$

where

$$M_i = \prod_{j=i+1}^{m} f_j, \qquad i = 1, \cdots m-1 \tag{4}$$

$$N_i = \sum_{p=i}^{m-1} \beta_p \left(\prod_{j=i+1}^{p} f_j \right), \qquad i = 1, \cdots, m-1 \tag{5}$$

Thus, from (3) we have $(m-1)$ linear equations with m variables, and together with the normalization equation, we have m equations. These equations can be solved to obtain the individual load fractions. Now, using (3) in the normalization equation, we obtain α_m as

$$\alpha_m = \left(\frac{L - X(m)}{Y(m)} \right) \tag{6}$$

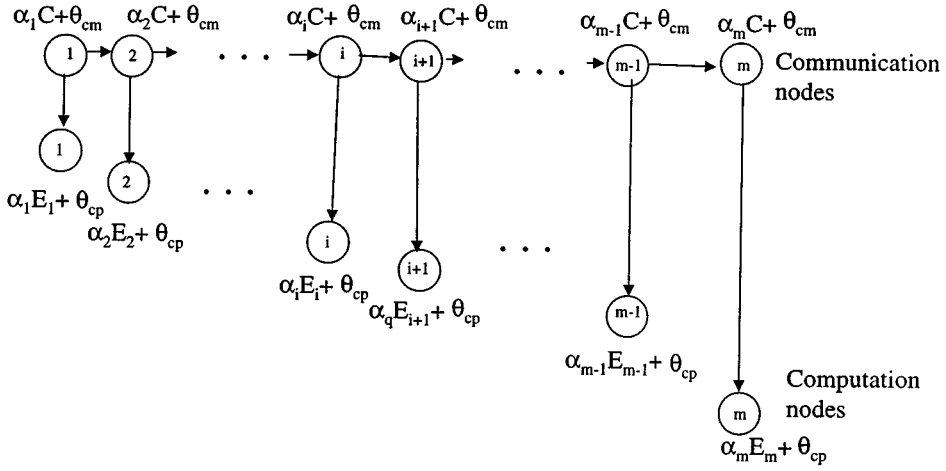


Fig. 2. Directed flow graph for load distribution in bus networks with m processors.

where

$$X(m) = \sum_{i=1}^{m-1} \sum_{p=i}^{m-1} \beta_p \left(\prod_{j=i+1}^{p} f_j \right), \tag{7}$$

$$Y(m) = \left(1 + \sum_{i=1}^{m-1} \prod_{j=i+1}^{m} f_j \right) \tag{8}$$

Substituting (6) in (3), we obtain the individual load fractions. From Fig. 2, we obtain the expression for the optimal path as

$$P^*(\alpha^*, m) = \alpha_1(C + E_1) + \theta_{cm} + \theta_{cp} \tag{9}$$

Using $i = 1$ in (3) and substituting in (9), we obtain

$$P^*(\alpha^*, m) = (\alpha_m M_1 + N_1)(C + E_1) + \theta_{cm} + \theta_{cp} \tag{10}$$

where α_m, M_1, and N_1 are as defined above.

Thus, in the above analysis, we have obtained an optimal solution involving m processors by solving the set of recursive equations as shown above. Following the above steps, we can derive the optimal processing time for a system of m processors with a BCU *without overheads* ($\theta_{cm} = \theta_{cp} = 0$). The optimal processing time in this case, is given by

$$P^*(\alpha', m) = \frac{M_1(C + E_1)}{Y(m)} \tag{11}$$

where M_1 and $Y(m)$ are as defined above.

It is worth mentioning at this juncture that given a m-processor system, with the inclusion of all the overheads, it may not be necessary that an optimal solution exists when one attempts to utilize all the m-processors. This behavior can be seen from Fig. 3(a). We have used $L = 1$ in generating these performance curves. Here, we observe that as we tend to increase the number of processors, the processing time decreases. Also, we have shown the influence of $X(m)$ [given by (7)], which we will refer to as the *overhead factor*, in Fig. 3(b). From this figure, we see that the overhead factor $X(m)$ also increases as m increases. Also, we observe that for the speed and overhead parameters chosen, the maximum number of processors that can be utilized with the given sequence of load distribution is $m^* = 14$. Thus, beyond this m^*, optimal solution ceases to exist. This is due to the fact that the value of the overhead factor becomes *greater than 1*, and hence, there will not be any gain in the time performance even if we attempt to utilize more processors beyond m^*. Therefore, as long as the value of $X(m)$ is less than L, we can utilize all the m processors to process the entire load in a minimum amount of time with this fixed sequence.

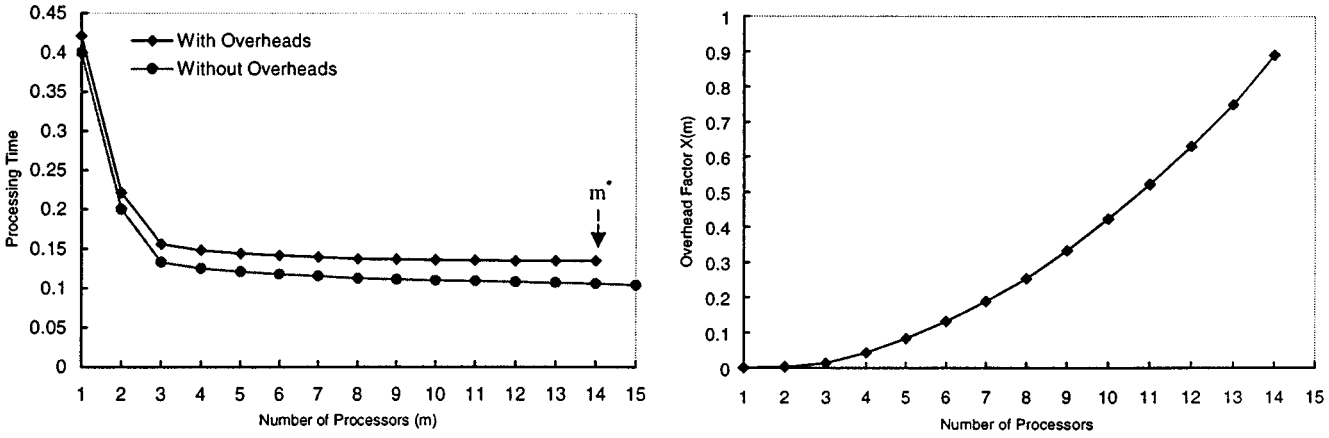
Thus, we see that a *necessary and sufficient* condition to obtain an optimal processing time using all the m-processors is given by

$$X(m) = \sum_{i=1}^{m-1} \sum_{p=i}^{m-1} \beta_p \left(\prod_{j=i+1}^{p} f_j \right) < L \tag{12}$$

The above condition is also referred to as *load sharing condition* in the literature. What we observe here is a restricted monotonic nature of the processing time behavior as opposed to the case when no overheads were considered in the problem formulation [7]. Given a fixed sequence, whenever an optimal solution using a k-processor system $p_1, \cdots, p_k$ ceases to exist, then we may utilize a maximal subset of k', $k' < k$ processors in the same sequence involving processors $p_1, \cdots, p_{k'}$ to obtain the optimal processing time. Of course, again we have to check the condition (12) for this set of k' processors.

Another observation that one could make from the Fig. 3(a) is on the rate at which the decrease in the time performance is achieved. Beyond $m = 7$ or $m = 8$, we see that the rate of decrease of the processing time is not significant and hence one can utilize $m = 7$ or 8 processors without having to utilize all the processors till $m^* = 14$. This in a way reduces the overhead processing by the system considerably, as it can be seen by the increase in the values of $X(m)$ for $m = 7$ and $m = 8$, respectively. In [20], [21], rigorous analysis on the influence of this overhead factor is carried out and the effect of load sequencing is also analyzed.

Remarks: As mentioned in Section I, from a practical perspective, a divisible load, in general, may not be truly arbitrarily divisible. An example of this scenario would be in computing a



Speed Parameters:

C	θ_{cp}	θ_{cm}	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}	E_{11}	E_{12}	E_{13}	E_{14}	E_{15}
0.1	0.02	0.001	0.3	0.2	0.1	0.4	0.6	0.7	0.8	0.5	0.9	1.1	1.3	1	0.6	0.5	0.3

Fig. 3. Behavior of the processing time and the overhead factor with respect to m (heterogeneous system).

large size matrix–vector product on parallel and distributed systems [12], [13] or in any image processing application [9]. For instance, in the above mentioned applications, usually the load that is assigned to a processor will be in terms of certain number of rows or columns. Hence, we can say that the load that is assigned will be an integral multiple of some fundamental quantity, referred to as the *divisibility factor* and is defined as the minimum possible *granularity* of any load fraction that can be assigned to a processor. This is denoted as δ. Further, we assume that the total load contains L units of δ load. In other words, the total load is $L\delta$, L being a variable and δ a constant. Without loss of generality, all the subsequent results presented in this paper assume $\delta = 1$.

III. Integer Approximation Algorithm for the Single Installment Strategy

In this section, for the strategy explained in Section II-D, we now propose an algorithm that generates integer load fractions and show that this algorithm guarantees a solution that lies within the acceptable limits of time performance when compared with the optimal solution. It may be recalled that the optimal solution obtained in Section II-D is by assuming that the load is arbitrarily divisible and that all the processors stop computing at the same time. The algorithm that is to be presented in this section restricts the division of the load to integer values. Hence, it is expected that once integer approximation is applied, all the processors will not stop computing at the same time instant. We propose an algorithm that is similar to rounding-off procedure in generating integer load fractions, however takes exactly $O(m)$ steps to converge. An interesting feature of the algorithm is that it keeps track of the amount of load that is scheduled so far and the amount of load that remains to be scheduled. This feature avoids the possibility of frequent back-tracking and aids in monitoring the time performance to be within the acceptable limits.

We refer to the optimal solution obtained in Section II-D as Opt_Sol. As a starting point of our algorithm, we shall assume that the initial load distribution, given by Opt_Sol, denoted as $\alpha = (\alpha_1, \cdots, \alpha_i, \cdots, \alpha_m)$. Let $\alpha'_1 = \lceil \alpha_1 \rceil$. Obviously, α'_1 is an integer. Let $\Delta\alpha_1 = (\alpha'_1 - \alpha_1)$ denote the difference in the values of the load fractions assigned to p_1. Obviously, $\Delta\alpha_1 < 1$. Next, we set $\alpha'_2 = \lceil \alpha_2 \rceil$ provided $\Delta\alpha_1 + \Delta\alpha_2 < 1$. If this condition fails to hold then, we set $\alpha'_2 = \lfloor \alpha_2 \rfloor$. This process is repeated for all the processors and results in a schedule $\alpha' = (\alpha'_1, \cdots, \alpha'_i, \cdots, \alpha'_m)$, where $\alpha'_i, i = 1, \cdots, m$, is an integer. The central idea of this algorithm lies in approximating the real values of the load fractions given by OS to integer values either by using the *smallest integer value that is greater than the current real value* or by using a *greatest integer smaller than the current real value* of the load fraction. This process is adopted at every iteration and the idea is either to "push" the excess load to or "accept" the excess load from the adjacent processors. The algorithm is described below.

```
Algorithm
α'_1 = ⌈α_1⌉;
sum_1 = α'_1 − α_1;
For (i = 2; i <= m; i = i + 1)
{
  If (sum_{i−1} + ⌈α_i⌉ − α_i < 1)
    α'_i = ⌈α_i⌉;
  else
    α'_i = ⌊α_i⌋;
  sum_i = sum_{i−1} + (α'_i − α_i);
{
```

In the above algorithm the sum_i represents the accumulated carry ($\sum_{j=1}^{i-1} sum_j + \Delta\alpha_i$), where $\Delta\alpha_i = (\alpha'_i - \alpha_i)$ in ith iteration. Example 1 presented at the end of next section illustrates the above procedure.

A. Performance Bounds

In this section, we shall derive some important properties that lead to the derivation of the performance bounds. One interesting fundamental issue to explore is the following. At the termination of the algorithm, whether the entire load gets processed or not, which shows the convergence property of the algorithm. We show below that the algorithm indeed guarantees this aspect.

Lemma 1: In the above algorithm, $-1 \leq sum_i \leq 1$, $i = 1, \cdots, m$.

Proof: The proof of the lemma is evident from the working style of the above algorithm. **Q.E.D**

Lemma 2: Let sum_i be defined as in the above algorithm. Then, $sum_m = \sum_{i=1}^{m} \Delta\alpha_i = 0$.

Proof: We prove this by contradiction. Suppose the above claim is not true. Then, this means that $sum_m = \sum_{i=1}^{m} sum_i = x$, and $x \neq 0$. Also, we know that $\sum_{i=1}^{m} \alpha_i = L$ and L is an integer. Rewriting this expression in terms of $\Delta\alpha_i$ and α_i', we obtain, $\sum_{i=1}^{m} \alpha_i' - sum_m = L$. This can be rewritten as, $\sum_{i=1}^{m} \alpha_i' - x = L$.

Since the first term on the LHS, and the term on the RHS are integers, x must also be an integer. From Lemma 1, we observe that $x = 0$, since $x = 0$ is the only integer that lies between -1 and $+1$, which contradicts our assumption, thus proving the lemma. **Q.E.D.**

The above lemma justifies the design of the algorithm by showing that at the completion of the algorithm, the entire load is processed.

Now, for a *homogeneous system*, we shall show that the processing time solution obtained by the above algorithm is no greater than Opt_Sol by an amount equal to the sum of communication and computation time of δ units of load. We assume that (12) is satisfied for this m-processor sequence, and hence all the m processors participate in processing the load.

Theorem 1: Consider a homogeneous bus network. Let $\alpha^* = (\alpha_1, \cdots, \alpha_i, \cdots, \alpha_m)$ be the optimal load distribution under infinite divisibility assumption that gives Opt_Sol. Let $\alpha' = (\alpha_1', \cdots, \alpha_i', \cdots, \alpha_m')$ be the load distribution generated by the above algorithm that results in a critical path $P(\alpha', m)$. Then, $P(\alpha', m) < P^*(\alpha^*, m) + (C + E)$.

Proof: Let $P_i(\alpha', m) = P(\alpha', m)$. From Fig. 2, we obtain

$$P_i(\alpha', m) = \sum_{j=1}^{i} \alpha_j' C + \alpha_i' E + i\theta_{cm} + \theta_{cp}. \tag{13}$$

Similarly, for α^* distribution, we obtain

$$P^*(\alpha^*, m) = P_i^*(\alpha^*, m) = \sum_{j=1}^{i} \alpha_j C + \alpha_i E + i\theta_{cm} + \theta_{cp}, \tag{14}$$

From (13) and (14), we obtain

$$P_i(\alpha', m) - P^*(\alpha^*, m) = \sum_{j=1}^{i} (\alpha_j' - \alpha_j) C + (\alpha_i' - \alpha_i) E, \tag{15}$$

$$\Rightarrow P_i(\alpha', m) - P^*(\alpha^*, m) = \sum_{j=1}^{i} \Delta\alpha_j C + \Delta\alpha_i E \tag{16}$$

From Lemma 1, we immediately observe

$$\sum_{j=1}^{i} \Delta\alpha_j C + \Delta\alpha_i E < (C + E), \tag{17}$$

$$\Rightarrow P_i(\alpha', m) = P(\alpha', m) < P^*(\alpha^*, m) + (C + E). \tag{18}$$

Hence the proof. **Q.E.D.**

The above theorem explains the "near-optimality" of the solution obtained by using the above algorithm to the solution generated under infinite divisibility assumption by an amount equal to the sum of the communication and computation times of the smallest unit δ load. The following example demonstrates the above algorithm and the theorem.

Example 1: Consider a homogeneous system consisting of $m = 5$ processors with the following parameters: $C = 1.0$, $E = 1.0$, $\theta_{cp} = 0.002$, and $\theta_{cm} = 0.001$. Let the size of the load be $L = 100$ units. Using (10), we obtain the optimal processing time is $P^*(\alpha^*, 5) = 103.231$, and load distribution $\alpha^* = (51.6, 25.8, 12.9, 6.5, 3.2)$. Using the integer approximation algorithm, we obtain the integer load distribution $\alpha' = (52, 26, 13, 6, 3)$. We observe that the results of Lemma 1 and Lemma 2 hold. From (13), for all $i = 1, 2, \cdots, 5$, we obtain the finish times of each processor as $104.021, 104.022, 104.023, 103.024,$ and 103.025, respectively. The processing time after integer approximation is given by, $P(\alpha', 5) = 104.023$. We see that $P(\alpha', 5) < P^*(\alpha^*, 5) + (C + E)$, thus verifying Theorem 1.

IV. Constrained Partitioning

In this section, we tackle another closely related problem of scheduling divisible loads. As mentioned in Section I, we consider the problem of scheduling a divisible load data under the constraint that the entire load cannot be divided into more than k partitions. Applications that fall into this kind of treatment mostly belong to image and computer vision data processing domain. One of the typical image processing applications that belongs to such class of problems addressed in this paper is the problem of human facial feature detection using edge counting [12]. This application demands that the entire image data cannot be divided into arbitrarily smaller fractions since the template used in the feature detection process counts the edge pixels into three distinct rectangular areas of the image simultaneously to find a global maximum. These three rectangular areas are separated by a distance that is dynamically altered according to the size of the image and the density of the edge pixels around the feature areas. The minimum size of each of the rectangular areas should be greater than the maximum possible size of a feature area (eye, mouth or nose) of the given facial image. Thus, the image demands a restricted partitioning

in and processing concurrently on different processors. The results also hold whenever any restriction on the level of partitioning is imposed. For instance, when granularity of the data becomes an important factor in processing the data on a distributed system, the approach presented in this paper becomes the natural choice and provides a systematic way to construct an optimal schedule. In reality, most of the times the selection of an appropriate grain size (granularity) during the runtime becomes an important factor, as the effective speed of the network and the processors may vary due to the other loading effects. We address the problem of scheduling such loads which impose restrictions in partitioning in a purely arbitrary fashion.

A. Load Distribution Strategy and Recursive Equations

Formally, we state the problem as follows. Let the BCU start to distribute the processing load from p_1 to p_m in at most $k \geq m$ installments, i.e., in k rounds of load distribution. More specifically, we pose the following question. *Given a distributed bus network with m homogeneous processors and also given that the maximum number of fractions k into which the processing load can be divided, what is the optimal load distribution that minimizes the processing time, by taking into account all the overhead components that penalize the time performance?*. Note that, here too, the sizes of the partitioned load fractions assume any value between $[0, L]$, where, L is the total amount of load. The entire load distribution process by the BCU is represented in the form of a **directed flow-graph**, as shown in Fig. 4. The load distribution process takes place in the following manner. Note that the schedule shown in Fig. 4, consists of k communication nodes. Let α_i denote the load fraction assigned to a processor $f(i) = i$, for $i \leq m$, and $(m \bmod i)$, for $i > m$ in the installment $g(i) = \lceil (i/m) \rceil$. Thus, when $k = 5$ and $m = 3$, p_1 is assigned α_1 and α_4 in two installments, p_2 is assigned α_2 and α_5 in two installments, and p_3 is assigned α_3 in the first installment. For this case, for the ease of understanding, we re-denote the finish time path i as $P_i(s(k))$. Note that the definition of the finish time path is the same, i.e., the sum of the weights of the nodes starting from communication node 1 till the last computation node through the communication node i along the directed arrows. Here, $s(k)$ denotes the load distribution with k nodes, and is given by, $s(k) = \{\alpha_1, \cdots, \alpha_k\}$. In order to obtain the optimal processing time, we equate all the finish time paths, i.e., $P_i(s(k)) = P_{i+1}(s(k))$, where $P_i(s(k))$ can be obtained from Fig. 4., as

$$P_i(s(k)) = \sum_{j=1}^{i} \alpha_j C + i\theta_c m + \sum_{j=i}^{r} \alpha_i E_{f(i)} + (r-i)\theta_{cp}. \quad (19)$$

Note that we have assumed that processor p_i receives r installments out of k, and hence the last computation node for p_i will be r. It may be noted that by equating $P_i(s(k)) = P_{i+1}(s(k))$, we obtain a set of $k-1$ equations with k unknowns. Now, using the fact that the total load is L units, we have a total of k equations. These equations may be solved to obtain all the individual load fractions as explained in the previous section. However, for the purpose of computational ease, the above set of path equations $P_i(s(k))$ can be represented in the matrix form. We refer to this matrix as *path matrix*. Thus, in matrix notation, for $k = 5$ and $m = 3$, we represent (19) as shown in the first equation at the bottom of the page where, the row i denotes the path $P_i(s(k))$. Now, by equating $P_i(s(k))$ and $P_{i+1}(s(k))$, as per the principle of optimality [7], we obtain $m-1$ recursive equations involving m variables. With our normalization equation, we can solve all the m equations to obtain the individual load fractions. Thus, the difference between $P_i(s(k))$ and $P_{i+1}(s(k))$ can be represented in a matrix form, referred to as a *difference matrix*. Thus, for the case $k = 5$ and $m = 3$ the difference matrix is given by the second equation shown at the bottom of the page. Note that the row i in the above $k \times k$ matrix is the result of $P_i(s(k)) - P_{i+1}(s(k))$ and the last row is the normalization equation. The $k \times k$ difference matrix can be generated for any arbitrary m and k values as described in the procedure presented in Table I.

It may be noted that the procedure presented in Table I will be computationally easy to generate the equations in a matrix form for a given m and k values. Further, the generated matrix can be reused when either k or m values are altered. Thus, the solution to our problem lies in simply finding the inverse of the

$$\begin{pmatrix} (C+E_1) & 0 & 0 & E_1 & 0 \\ C & (C+E_2) & 0 & 0 & E_2 \\ C & C & (C+E_3)0 & 0 & \\ C & C & C & (C+E_1) & 0 \\ C & C & C & C & (C+E_2) \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \end{pmatrix} + \begin{pmatrix} \theta_{cm} + 2\theta_{cp} \\ 2\theta_{cm} + 2\theta_{cp} \\ 3\theta_{cm} + \theta_{cp} \\ 4\theta_{cm} + \theta_{cp} \\ 5\theta_{cm} + \theta_{cp} \end{pmatrix}$$

$$\begin{pmatrix} E_1 & -(C+E_2) & 0 & E_1 & -E_2 \\ 0 & E_2 & -(C+E_3) & 0 & E_2 \\ 0 & 0 & E_3 & -(C+E_1) & 0 \\ 0 & 0 & 0 & E_1 & -(C+E_2) \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \end{pmatrix} = \begin{pmatrix} \theta_{cm} \\ \theta_{cm} \\ \theta_{cm} \\ \theta_{cm} \\ L \end{pmatrix}$$

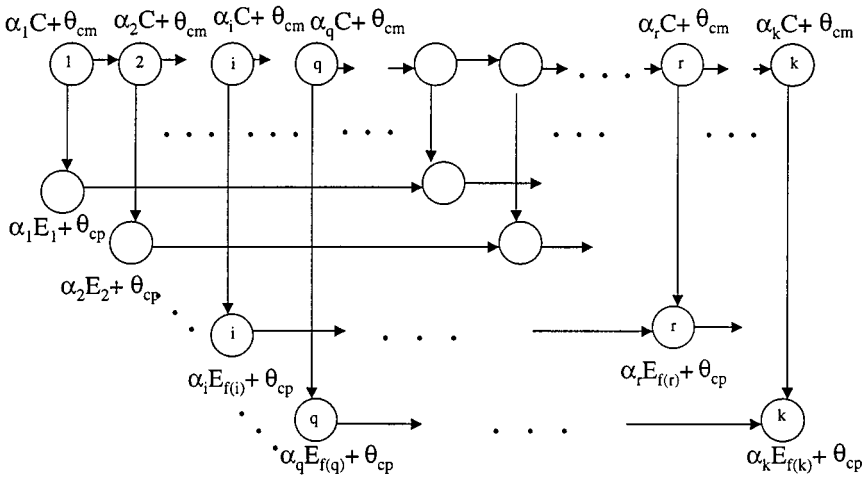


Fig. 4. Directed flow graph for load distribution in bus networks with k communication nodes.

TABLE I
$k \times k$ DIFFERENCE MATRIX GENERATION PROCEDURE

The $k \times k$ difference matrix

For $i \leq m-1$,

$x_{i,i} = E_i$
$x_{i,1} = \cdots = x_{i,i-1} = 0$
$x_{i,i+1} = -(C + E_{i+1})$
$x_{i,j} = 0, \forall j \neq i + nm$
$x_{i,j} = E_i, \forall j = i + nm$
$x_{i,j} = -E_{i+1}, \forall j = i + nm + 1,\ n = 1, 2, ...,\ j < k$

Define $q(i) = i \bmod m,\ i \neq nm$, and $q(i) = m, i = nm,\ n = 1, 2...,\ j < k$.

For $m \leq i < k$,

$x_{i,i} = E_{q(i)}$
$x_{i,1} = \cdots = x_{i,i-1} = 0$
$x_{i,i+1} = -(C + E_{(i \bmod m)+1})$
$x_{i,j} = 0, \forall j \neq i + nm, j < k$
$x_{i,j} = E_{q(i)}, \forall j = i + nm,\ j < k$
$x_{i,j} = -E_{(i \bmod m)+1}, \forall j = i + nm + 1,\ n = 1, 2, ...,\ j < k$

For $i = k$, $x_{k,j} = 1,\ \forall j = 1, ..., k$

The RHS column vector of the differcen matrix is given by,
$x_{i,1} = \theta_{cm},\ \forall i = 1, ..., k-1,$ and $x_{k,1} = L$.

difference matrix and multiplying with the transpose of the RHS column vector. However, it may be noted that it is quite possible that these equations may not yield feasible values (nonnegative values) for the load fractions. This means that these equations are not solvable to yield an optimal solution with k partitions and using m processors in the system. Thus, if an optimal solution does not exists for this particular value of k, then a lower value of k is to be attempted by following the above mentioned recursive procedure. However, note that the difference matrix for lower values of k can be extracted directly from the already generated $k \times k$ matrix.

Example 2: Consider a homogeneous network consisting of three processors. Let $E = 2.5$, $C = 0.95$, $\theta_{cm} = 0.01$, and $\theta_{cp} = 0.01$. Also, let $L = 2048$. Further, let the maximum allowed number of partitions be $k_{\max} = 5$. This is a typical example of an image processing application in which we consider an image of size 2048×2048, and partition the entire image in terms of number of rows. Hence, we let $L = 2048$. Following the above mentioned procedure, we obtain the optimal processing time as 2603.8 units, and the optimal load distribution in given by, (491.4592, 428.6214, 501.3595, 363.3011, 263.2588), respectively. Thus, for this case any lower value of k will only result in more processing time.

However, suppose if the processing speed of the processors are much faster, say $E = 0.01$, then following the above procedure, we see that for $k = 5$ and 4, the optimal solution does not exist, as some of the values of the load fractions are infeasible. Thus, the maximum allowed number of partitions is $k = 3$, for which the processing time is given by, 2048 units, and the optimal load distribution is given by, (2027.7, 20.1, 0.2), respectively. This result is of course intuitively true as the processing speeds are much faster than the communication speeds, and the gain achieved in sharing the total load with more than one processor is very small.

It may be observed that the difference matrix generated for $k = 5$ case can be reused for $k = 3$ case, by considering the first two rows and three columns of $k = 5$ case and appending a string of ones as the last (third) row. Further, it may be noted that the values we have chosen for the parameters are purely arbitrary and this choice was just for the purpose of demonstration.

A very important observation to make at this juncture is that, in general, it may be possible that an optimal solution may not exist for $k = k_{\max}$. This is mainly due to the combined effect of the overhead components becoming extremely large, resulting in infeasible values. Thus, the best possible solution may not be due to partitioning the load into $k_{\max}$ installments.

V. SUBOPTIMAL PROCESSING TIME SOLUTIONS

Based on the optimal processing time mentioned in the previous section, first, we shall propose two algorithms namely, **PIA** and **IIA** that generate integer valued load fractions to individual processors using the optimal solution obtained in the previous section. Both the algorithms use the optimal load distribution as the initial schedule and this means, the number of communication nodes used in the following algorithms remains same as that of the optimal solution. Let us denote this optimal solution as Opt_Sol. We present two different types of algorithms carrying out integer approximations.

A. Algorithm PIA

In this algorithm, referred to as PIA (Processor based Integer Approximation), we carry out the integer approximation at each installment for every processor starting from its first installment to its last installment. In doing so, we accumulate the "residue" or the "carry" generated in each iteration and propagate it till node k is reached. In other words, an excess load (as defined in the algorithm) at an installment $g(i)$ at the processor $f(i)$ is carried to the same processor in the immediate next installment. For the ease of developing this method, we now adopt the following notations. Let each processor p_i be assigned n_i number of installments, where $n_i \leq k$, such that $\sum_{j=1}^{m} n_j = k$. Then, we denote the load fraction assigned to processor p_i in jth installment, $j \leq n_i$ as $\alpha_{i,j}$. Note that these n_i, $i = 1, \cdots, m$ are obtained from the optimal schedule generated after solving the difference matrix described in the previous section.

```
Algorithm PIA
sum_1 = 0; i = 1;
do {
  For j = 1 to n_i {
    if (sum_i + ⌈α_{i,j}⌉ − α_{i,j}) ≤ 0.5
    then α'_{i,j} = ⌈α_{i,j}⌉;
    else α'_{i,j} = ⌊α_{i,j}⌋;
  sum_i = sum_i + α'_{i,j} − α_{i,j};
  }
  i = i + 1;
  sum_i = sum_{i−1}; }
  while (i ≤ m);
```

In the above algorithm the sum_i is an accumulated carry at any (i, j)th iteration. At this juncture, an interesting point to verify is whether the entire load gets processed at the completion of the algorithm or not. We show below that the algorithm indeed guarantees this aspect.

Lemma 3: At the end of any iteration (i, j), $-0.5 < sum_i \leq 0.5$.

Proof: We shall prove this by mathematical induction. When $(i, j) = (1, 1)$, the above assertion holds trivially. As an induction hypothesis step, suppose it holds at (i, j). Further, let us denote the sum_i at (i, j)th iteration as $sum_{i,j}$, and let $\Delta\alpha_{i,j} = \alpha'_{i,j} - \alpha_{i,j}$. From the algorithm, the expression for the sum at step (i, j) is given by

$$sum = \sum_{l=1}^{i-1} \sum_{k=1}^{n_i} \Delta\alpha_{l,k} + \sum_{k=1}^{j} \Delta\alpha_{i,k}, \qquad j \leq n_i$$

and by the above hypothesis, we have, $-0.5 \leq sum_{i,j} \leq 0.5$. Now, we have the following two cases to consider.

i) If $sum_{i,j} + \lceil\alpha_{i,j+1}\rceil - \alpha_{i,j+1} \leq 0.5$ then, $\alpha'_{i,j+1} = \lceil\alpha_{i,j+1}\rceil$ and $\Delta\alpha_{i,j+1} = \lceil\alpha_{i,j+1}\rceil - \alpha_{i,j+1}$. This means, $-0.5 < sum_{i,j+1} = sum_{i,j} + \Delta\alpha_{i,j+1} \leq 0.5$.

ii) If $sum_{i,j} + \lceil\alpha_{i,j+1}\rceil - \alpha_{i,j+1} > 0.5$ then, $\alpha'_{i,j+1} = \lfloor\alpha_{i,j+1}\rfloor$ and $\Delta\alpha_{i,j+1} = \lceil\alpha_{i,j+1}\rceil - \alpha_{i,j+1} - 1$. This means, $-0.5 < sum_{i,j} = sum_{i,j+1} + \Delta\alpha_{i,j+1} \leq 0.5$.

Since every iteration is governed by two indices, we have to prove the lemma when i varies, i.e., suppose the lemma holds at (i, n_i), then we have to prove for $(i + 1, 1)$. The proof of this is similar to case ii) shown above, and we omit details. **Q.E.D.**

Lemma 4: $\sum_{i=1}^{m} \sum_{j=1}^{n_i} \Delta\alpha_{i,j} = 0$.

Proof: We shall prove this by contradiction. Suppose otherwise. This means

$$\sum_{i=1}^{m} \sum_{j=1}^{n_i} \Delta\alpha_{i,j} = x, \qquad x \neq 0,$$

Also, we know that $\sum_{i=1}^{m} \sum_{j=1}^{n_i} \Delta\alpha_{i,j} = L$, L is an integer. Rewriting the above expression in terms of $\Delta\alpha_{i,j}$ and $\alpha'_{i,j}$ we obtain, $\sum_{i=1}^{m} \sum_{j=1}^{n_i} \alpha'_{i,j} - x = L$. Since the first term on the LHS and the term on the RHS are integers, x must also be an integer. From Lemma 3, we observe that $x = 0$ as the only integer between -0.5 and 0.5, thus contradicting our assumption. Hence, the proof. **Q.E.D.**

This lemma justifies the fact that there is no load left without processing.

Lemma 5: For any $i = 1, \cdots, m$ $\sum_{k=1}^{n_i} \Delta\alpha_{i,k} < 1$.

Proof: From Lemma 3,

$$-0.5 < \sum_{l=1}^{i-1} \sum_{k=1}^{n_i} \Delta\alpha_{l,k} + \sum_{k=1}^{j-1} \Delta\alpha_{i,k}$$

and

$$\sum_{l=1}^{i-1} \sum_{k=1}^{n_l} \Delta\alpha_{l,k} + \sum_{k=1}^{j-1} \Delta\alpha_{i,k} \leq 0.5$$

Subtracting first inequality from the second, we prove the lemma. **Q.E.D.**

We use this result in the following theorem.

Now, for a homogeneous system, we shall show that the above algorithm yields a solution that is no greater than the OS by an amount $((mC/2) + E)$.

Theorem 2: Consider a homogeneous system of processors. Let $s^*(n_m) = (\alpha_{1,1}, \cdots, \alpha_{i,j}, \cdots, \alpha_{m,n_m})$ be the *Opt_Sol* under infinite divisibility assumption that gives the optimal solution, denoted as $P^*(s^*)$. Let $s' = (\alpha'_{1,1}, \cdots, \alpha'_{i,j} \cdots, \alpha'_{m,n_m})$ be the schedule generated by Algorithm PIA and let the processing time be $P(s'(n_m))$. Then, $P(s'(n_m)) < P^*(s^*(n_m)) + ((mC/2) + E)$.

Proof: For s^* and s' schedules, the difference in the processing times can be obtained from Fig. 4 as,

$$\begin{aligned} &P(s'(n_m)) - P^*(s^*(n_m)) \\ &\quad = \max_{1 \leq i \leq m,\, 1 \leq j \leq n_i} \left(\sum_{l=1}^{m} \sum_{k=1}^{j-1} \Delta\alpha_{l,k} + \sum_{l=1}^{i} \Delta\alpha_{l,j} \right) C \\ &\qquad + \sum_{k=j}^{n_i} \Delta\alpha_{i,k} E, \end{aligned} \tag{20}$$

since, by the definition of $\Delta\alpha_{i,j}$. Using Lemma 3 and Lemma 5, the result follows. **Q.E.D.**

It may be noted that the bound obtained in this case depends only on the system parameters E and C and on the number of processors m.

B. Algorithm IIA

In this algorithm, referred to as IIA (Installment based Integer Approximation), we carry out the integer approximation starting from node 1 to k (see Fig. 4) in the order in which these load fractions are distributed. In other words, the "excess" load at a particular processor $f(i)$ is carried to its successor in the order of the load distribution. Here too, we assume that the schedule with infinite divisibility assumption, is our initial distribution of the load fractions. Since this schedule is known *a priori*, we denote the number of processors involved in jth installment, $j \leq \max_i\{n_i\}, i = 1, \cdots, m$ as N_j. We now present the algorithm.

```
Algorithm IIA
sum_1 = 0, count = 1, j = 1;
  do {
    for i = 1 to N_j {
    if (sum_j + ⌈α_{i,j}⌉ − α_{i,j}) ≤ 0.5
    then α'_{i,j} := ⌈α_{i,j}⌉
    else α'_{i,j} := ⌊α_{i,j}⌋
    sum_j := sum_j + α'_{i,j} − α_{i,j};
    count = count + 1; }
  j = j + 1;
    sum_j = sum_{j−1}; }
while (count ≤ k);
```

In the above algorithm the sum_j is an accumulated carry at (i, j)th iteration. It may be noted that the lemmas 3 and 4 hold true for this case.

Lemma 6: $\sum_{l=j}^{n_i} \Delta\alpha_{i,l} < k/2$.

Proof: Result follows by applying Lemma 5, and noting that each $\Delta\alpha_{i,l} \leq 0.5, j, \cdots, n_i$. **Q.E.D.**

Now, for a homogeneous system, we shall show that the above algorithm yields a solution that is no greater than the OS by an amount $(C + (kE/2))$.

Theorem 3: Consider a homogeneous system of processors. Let $s^*(n_m) = (\alpha_{1,1}, \cdots, \alpha_{i,j}, \cdots, \alpha_{m,n_m})$ be the Opt_Sol under infinite divisibility assumption that gives the optimal solution, denoted as $P^*(s^*(n_m))$. Let $s'(n_m) = (\alpha'_{1,1}, \cdots, \alpha'_{i,j} \cdots, \alpha'_{m,n_m})$ be the schedule generated by the Algorithm IIA and let the processing time be $P(s'(n_m))$. Then, $P(s'(n_m)) < P^*(s^*(n_m)) + C + (kE/2)$.

Proof: The result follows immediately by using similar line of arguments presented in Theorem 2, however, we use Lemmas 3 and 6 in this case. We omit the details. **Q.E.D.**

It may be noted that this bound depends only on the parameters E, C and k and not on m. The following illustrative example describes all the above results.

Example 3: Consider the first part of Example 2. The optimal processing time is given by, $P^*(s^*(5)) = 2603.8$. We apply the algorithm PIA to obtain the integer load distribution as $(491, 428, 501, 364, 264)$, and the processing time of the entire load is given by, 2605.65, contributed by the finish time path $P_5(s(5))$ (by processor p_2). By Theorem 2, we see that $P(s'(n_m)) = 2605.65 < P^*(s^*(n_m)) + ((mC/2) + E) = 2607.725$.

When we apply the algorithm IIA, we obtain the integer load distribution as, $(491, 429, 501, 364, 263)$, and the processing time of the entire load is given by, 2605.8, contributed by the finish time path $P_4(s(5))$ (by processor p_1). By Theorem 3, we see that $P(s'(n_m)) = 2605.8 < P^*(s^*(n_m)) + C + (kE/2) = 2611.0$. Also, the results of all the lemmas can be immediately verified.

Following theorem suggests a way to make a choice of the above algorithms.

Theorem 4: If $C > (E(k/2 - 1)/(m/2 - 1))$, then $P_{IIA}(s(k) < P_{PIA}(s(k)$, where $P_{IIA}(s(k)$ and $P_{PIA}(s(k)$ are the suboptimal solutions generated by the respective algorithms.

Proof: The proof follows immediately by obtaining the difference between the suboptimal solutions generated by the algorithms given by Theorems 2 and 3 and using the condition given in the theorem. **Q.E.D**

Thus, in our Example 3 above, we see that $C = 0.95 < (E(k/2-1)/(m/2-1)) = 12.5$. Hence, for this system, algorithm PIA gives more tighter bound on the time performance.

VI. Discussion of the Results

The research contributions in this paper are significant to this domain of DLT. This paper attempts to bridge the gap between theory and practice by tuning the model and demonstrating the performance of the strategies under realistic situations. Since the effect of overhead components is embedded in the closed-form solution for the processing time, the decision on the choice on the number of processors to be utilized under heavily and lightly loaded network conditions can be made immediately. Fig. 3 is a good example to demonstrate this fact. Secondly, any practical implementation, based on the strategies proposed in the literature in DLT, demands quantum load dissemination in terms of either number of rows or columns, need to generate integer load distribution arises. It may be noted that the results of this paper remain unaffected whether column or row-wise striping is carried out in load distribution process. In Example 2, we carry out row striping in partitioning the load. In Section III, we proposed an integer approximation algorithm to generate these integer valued loads in a systematic way such that the suboptimal solution that is recommended by this algorithm can be very well quantified. Through this procedure, we have obtained an ultimate performance bound on the solution generated by this algorithm.

One of the useful contributions is the *directed flow graph* representation. With the aid of such representation, for instance, a multi-installment strategy [20], [21], [7] discussed in the literature can be easily be constructed and recursive equations can be generated by simply writing the path equations. We believe that this representation is elegant in capturing all the features of the timing diagram and also serves as a generalized representation to construct and analyze complex scheduling strategies in this domain.

We also tackle a closely related problem, but highly useful for a specific class of applications in image and computer vision processing areas. The results also hold whenever any restriction on the level of partitioning is imposed. For instance, when granularity of the data becomes an important factor in processing the data on a distributed system, the approach presented in this paper becomes the natural choice and provides a systematic way to construct an optimal schedule. In reality, most of the times the selection of an appropriate grain size (granularity) during the runtime becomes an important factor, as the effective speed of the network and the processors may vary due to the other loading effects. As a consequence, the model must be tuned to take care of this size dependent behavior. Since from practical perspective, generating integer load fractions is mandatory, we propose two different algorithms to generate integer load fractions namely, IIA and PIA. The algorithms are shown to converge and guarantee that the time performance to be well within acceptable limits. The choice of these algorithms, as shown in Theorem 4, is very important for time critical applications, as the suboptimal solution is influenced by the speed parameters of the system and the allowed number of partitions. The integer approximation algorithm proposed in Section 3 has the complexity $O(m)$. The complexity of the algorithms PIA and IIA is $O(k)$. As a final remark, we would like to point out the following. In reality, the quantities E and C are random, and we can only discuss the performance with respect to the average values of these quantities. The average values can be either estimated or obtained as a result of an empirical study. If we consider the average values for these quantities in Theorems 2–4, then these results are much more meaningful in giving an estimate of the suboptimal processing time solution under average performance.

VII. Conclusions

The problem of scheduling divisible loads on bus networks is considered. The research contributions in this paper have attempted to bring the theory closer to practice by employing a realistic model and analyzing the performance under practical conditions. This paper has the following major contributions. A directed flow graph representation was introduced to describe the load distribution process. This flow-graph representation is elegant in describing all the features of the load distribution strategy and can be very well altered depending upon the strategy. The mathematical model considered in this paper included all the overhead components that penalize the time performance in determining the optimal solution. Using this model, we have derived a closed-form solution for the optimal processing time. The closed-form solution clearly reflects the influence of the overheads and gives rise to an important necessary and sufficient condition for the existence of the optimal processing time by employing all the processors in the network. We have demonstrated the influence of these overhead factors on the time performance and its importance in the choice of number of processors that can be utilized in the network. We extended our study to suit a special class of problems in which partitioning of the data is restricted to a finite number. We have proposed a systematic way to generate the optimal solution.

From practical perspective, we have proposed different integer approximation algorithms that generate suboptimal solution from the optimal solution obtained through earlier analysis. The behavior of these algorithms are analyzed and for the class of homogeneous networks, we have derived the ultimate performance bounds. These bounds serve as an excellent estimate of the time performance of the system under different network conditions and indicate how far the solution(in practice) lie when compared to the ideal optimal solution. As mentioned in Section VI, these solutions may very well reflect the average time performance when the speed parameters are replaced by their average values.

Extensions to this research can be devoted to analyze the behavior of the strategies and the integer approximation algorithms on different architectures. It would be interesting to derive similar performance bounds for these architectures. Also, the effect of solution back propagation, which is beyond the scope of this paper, can provide a complete understanding of the behavior of the system.

Acknowledgment

The authors would like to thank D. Ligang for his help in preparing the figures in the manuscript.

References

[1] J. A. Stankovic, "Implications of classical scheduling results for real-time systems," *IEEE Comput. Mag.*, pp. 16–25, June 1995.

[2] Y. C. Cheng and T. G. Robertazzi, "Distributed computation with communication delays," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 24, pp. 700–712, 1988.

[3] V. Bharadwaj, D. Ghose, and V. Mani, "Optimal sequencing and arrangement in distributed single-level networks with communication delays," *IEEE Trans. Parallel Distrib. Syst.*, vol. 5, pp. 968–976, 1994.

[4] T. G. Robertazzi, "Processor equivalence for a linear daisy chain of load sharing processors," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 29, pp. 1216–1221, 1993.

[5] J. Blazewicz and M. Drozdowski, "The performance limits of a two-dimensional network of load sharing processors," Inst. Comput. Sci., Poznan Univ. Technol., Poznan, Poland, Res. Rep. RA-94/007, Nov. 1994.

[6] J. Blazewicz and M. Drozdowski, "Distributed processing of divisible jobs with communication startup costs," *Discr. Appl. Math.*, vol. 76, no. 1–3, pp. 21–41, 1997.

[7] V. Bharadwaj, D. Ghose, V. Mani, and T. G. Robertazzi, *Scheduling Divisible Loads in Parallel and Distributed Systems*. Los Almitos, CA: IEEE Comput. Soc. Press, Aug. 1996.

[8] D. Ghose and H.-J. Kim, "Load partitioning and trade-off study for large matrix-vector computations in multicast bus networks with communication delays," *J. Parallel Distrib. Comput.*, vol. 54, 1998.

[9] C. Lee, Y.-F. Wang, and T. Yang, "Static global scheduling for optimal computer vission and image processing operations on distributed-memory multiprocesors," Univ. California, Santa Barbara, TRC 94-23, Dec. 1994.

[10] G. D. Barlas, "Collection-aware optimum sequencing of operations and closed-form solutions for the distribution of a divisible load on arbitrary processor trees," *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, no. 5, pp. 429–441, May 1998.

[11] V. Bharadwaj, D. Ghose, and V. Mani, "Multi-installment load distribution in tree networks with delays," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 31, pp. 555–567, 1995.

[12] L. C. De Silva, K. Aizawa, and M. Hatori, "Detection and tracking of facial features by using a facial feature model and deformable circular templates," *IEICE Trans. Inform. Syst.*, vol. E78-D, no. 9, pp. 1195–1207, Sept. 1995.

[13] S. Sahni and V. Thanvantri, "Performance metrics: Keeping the focus on runtime," *IEEE Parallel Distrib. Technol.*, pp. 43–56, Spring 1996.
[14] D. Ghose and V. Mani, "Distributed computation with communication delays: Asymptotic performance analysis," *J. Parallel Distrib. Comput.*, vol. 23, no. 3, pp. 293–305, 1994.
[15] J. Sohn, T. G. Robertazzi, and S. Luryi, "Optimizing computing costs using divisible load analysis," *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, no. 3, pp. 225–234, Mar. 1998.
[16] S. Bataineh and M. Al-Ibrahim, "Effect of fault-tolerance and communication delay on response time in a multiprocessor system with a bus topology," *Comput. Commun.*, vol. 17, pp. 843–851, 1994.
[17] V. Mani and D. Ghose, "Distributed computation in linear networks: Closed-form solutions," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 30, pp. 471–483, 1994.
[18] V. Bharadwaj, H. F. Li, and T. Radhakrishnan, "Scheduling divisible loads in bus networks with arbitrary processor release times," in *Computers and Mathematics with Applications*. New York: Pergamon, 1996, vol. 32, pp. 57–77.
[19] J. Sohn and T. G. Robertazzi, "An optimum load sharing strategy for divisible jobs with time-varying processor speed and channel speed," in *Proc. ISCA Int. Conf. Parallel and Distributed Computing Systems*, Orlando, FL, September 1995, pp. 27–32.
[20] V. Bharadwaj, X. Li, and C. C. Ko, "On the influence of start-up costs in scheduling divisible loads on bus networks," IEEE Trans. Parallel Distrib. Syst., 2000, to be published.
[21] V. Bharadwaj, X. Li, and C. C. Ko, "On the influence of start-up costs in scheduling divisible loads on bus networks," Dept. Elect. Comput. Eng., Nat. Univ. Singapore, Rep. VB/DLT/001/1998, Oct. 1998.

Bharadwaj Veeravalli (M'99) received the B.Sc. degree in physics from Madurai-Kamaraj University, India, in 1987, and the M.S. degree in electrical communication engineering in 1991 and the Ph.D. degree from Department of Aerospace Engineering in 1994, both from the Indian Institute of Science, Bangalore.

He was a Post-Doctoral Research Fellow with the Department of Computer Science, Concordia University, Montreal, PQ, Canada, in 1996. He is currently with the Department of Electrical and Computer Engineering, The National University of Singapore, as an Assistant Professor His research interests include high speed heterogeneous computing, scheduling in parallel and distributed systems, and multimedia computing. He is one of the earliest researchers in the field of divisible load theory and is the principal author of the book *Scheduling Divisible Loads in Parallel and Distributed Systems* (Los Alamitos, CA: IEEE Computer Society Press, 1996).

N. Viswanadham (F'93) is a Professor in The Logistics Institute-Asia Pacific at the National University of Singapore, Singapore. He is the author of several text books, journal articles and conference papers. He is the lead author of two textbooks, *Reliability in Computer and Control Systems* (Amsterdam, The Netherlands, North-Holland, 1987) and *Performance Modeling of Automated Manufacturing Systems* (Englewood Cliffs, NJ: Prentice Hall, 1992), and is the author of *Analysis of Manufacturing Enterprises—An Approach to Leveraging the Value Delivery Processes for Competitive Advantage* (Norwell, MA: Kluwer). He is also the editor (or co-editor) of six other edited volumes. He is an associate editor of the *Journal of Manufacturing Systems*, I*ntelligent and Robotic Systems*, and *The Journal of Franklin Institute*. His current research interests include supply chain management and business to business commerce.

Dr. Viswanadham is a Fellow of the Indian National Science Academy, Indian Academy of Sciences, Indian National Academy of Engineering, and Third World Academy of Sciences. He is currently a senior editor of the IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION and an associate editor of the IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS,