Adaptive Divisible Load Scheduling Strategies for Workstation Clusters with Unknown Network Resources

Debasish Ghose, *Member*, *IEEE*, Hyoung Joong Kim, *Member*, *IEEE*, and Tae Hoon Kim

Abstract—Conventional divisible load scheduling algorithms attempt to achieve optimal partitioning of massive loads to be distributed among processors in a distributed computing system in the presence of communication delays in the network. However, these algorithms depend strongly upon the assumption of prior knowledge of network parameters and cannot handle variations or lack of information about these parameters. In this paper, we present an adaptive strategy that estimates network parameter values using a probing technique and uses them to obtain optimal load partitioning. Three algorithms, based on the same strategy, are presented in the paper, incorporating the ability to cope with unknown network parameters. Several illustrative numerical examples are given. Finally, we implement the adaptive algorithms on an actual network of processor nodes using MPI implementation and demonstrate the feasibility of the adaptive approach.

Index Terms—Scheduling and task partitioning, divisible loads, distributed applications, workstations, multiprocessor systems.

1 Introduction

The divisible load scheduling literature customarily deals with the processing of massive loads that can be partitioned into smaller load fractions and distributed to processing nodes over a network. The communication delays occurring in the network are explicitly accounted for in the model. Research in this area was initiated by two papers by Cheng and Robertazzi [1], [2]. A paper that independently introduced several concepts related to processing of divisible loads is by Agrawal and Jagadish [3]. Since then, two monographs [4], [5] and two recent survey/tutorial papers [6], [7] have been published on this topic apart from several papers that address the problem of distributing loads under practical constraints such as different release times, constant communication start-up times, load distribution under time-varying network capacity, different speed memory hierarchy, different architectures, performance limit analysis and multiple loads, and applications such as matrix-vector computation problems, query processing, image processing, and cost optimization [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16].

In general, in the conventional divisible load scheduling literature, for a given network topology, if z represents the measure of communication delay in the network and w the processing time for a unit load (z and w could be vectors where each element represents the corresponding parameter for the processor node and the communication link to it), then the load fraction α of the total load sent to each processor node is expressed as

$$\alpha = f(w, z). \tag{1}$$

Here, α is a vector with element α_i as the load fraction allocated to the ith processor node. Much of divisible load scheduling literature concerns itself with variations on this basic theme in terms of examining the effect of different architectures, using realistic models for communication delay and processing time calculation, asymptotic performance analysis, load distribution conditions, and so on. This has yielded a rich set of results in the divisible load scheduling literature over the past decade. The analytical results obtained are mainly based on a Gantt-chart-like timing diagram that represents the process of communicating and processing a load fraction by a processing node.

However, most of these treatments are based on assumptions of known and deterministic network resources in terms of the values of the parameters w and z. Lately, researchers have started addressing issues that are application oriented and go beyond these simple models of load distribution. This paper has a similar motivation and presents an adaptive load distribution strategy that does not depend on the prior knowledge of these parameters, but uses a probing technique to obtain their values as a part of the overall load distribution strategy. We believe that this approach opens up new possibilities for application-oriented divisible load scheduling research.

2 Basic System Architecture and Notations

Divisible load scheduling literature is replete with results and experiments that have been carried out on networks that are represented as single-level tree architectures. It is usually presumed that a single-level tree architecture represents only a bus-based system, such as the one shown in Fig. 1. But, the single-level tree architecture is a generic

- *D. Ghose is with the Guidance, Control, and Decision Systems Laboratory, Department of Aerospace Engineering, Indian Institute of Science, Bangalore 560 012, India. E-mail: dghose@aero.iisc.ernet.in.*
- *H.J. Kim and T.H. Kim are with the Department of Control & Instrumentation Engineering, Kangwon National University, Chuncheon 200-701, South Korea. E-mail: {khj, hoon}@kangwon.ac.kr.*

Manuscript received 26 Dec. 2003; revised 6 July 2004; accepted 6 Nov. 2004; published online 22 Aug. 2005.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0239-1203.

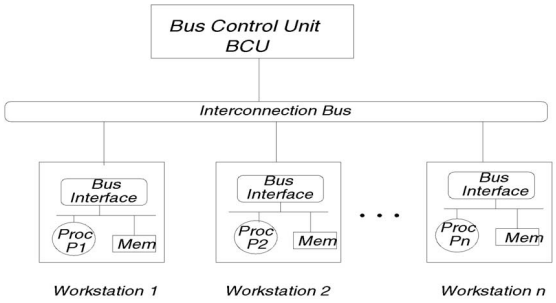


Fig. 1. A bus-based workstation cluster.

model for divisible load scheduling for a wide variety of distributed networks with different network topologies. For instance, Fig. 2a shows a general topology with the load distribution sequence (given by numbers) from the control processor to different processing nodes or workstations. Fig. 2b shows a multilevel bus topology where a similar load distribution occurs. It is assumed that communication of load to the ith processing node has to be complete before the communication of load to the $(i+1)$th processing node commences. This gives rise to a single-level heterogeneous tree network with different communication bandwidths and processing nodes with different speeds, depending on the available processor capacity at a node. Fig. 3 represents the generic model of such a general architecture.

Below, we define some parameters used in the load distribution analysis:

$z_i =$ Ratio of the time taken by a communication link i, to the time taken by a reference link, to communicate a given amount of load (thus, z is inversely proportional to the speed of the communication link).

$w_i =$ Ratio of the time taken by a processing node or workstation P_i, to the time taken by a reference processor, to compute a given amount of load (thus, w is inversely proportional to the speed of a processing node).

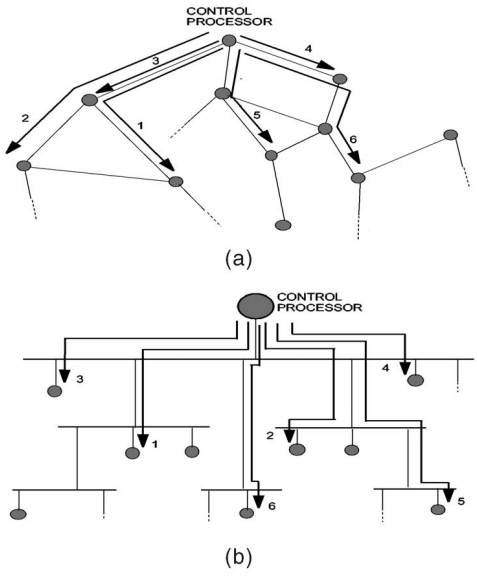


Fig. 2. General network topology.

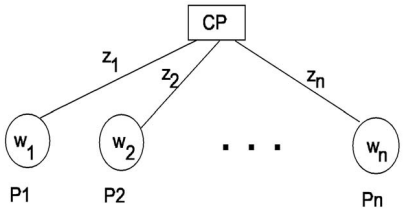


Fig. 3. A general single-level tree architecture.

$T_{cm} =$ Time taken by a reference link to communicate one unit of load.

$T_{cp} =$ Time taken by a reference processing node to process or compute one unit of load.

$L =$ The total load to be distributed and processed.

$\alpha_i =$ The fraction of the load assigned to a processing node Pi, with $\alpha_i \in [0,1]$.

$\eta =$ The fraction of the total load used for the probe phase. The total load is a multiple of this fraction and ηL is called an *installment*.

The following assumptions are made in formulating the algorithm. Most of these are reasonable and some of them simplify the process of load distribution without compromising the utility of the proposed algorithms:

1. Communication start-up time is negligible.
2. Time for passing messages is negligible (or they may be absorbed in the link speed model) compared to the time taken for communication and processing of computational loads at the nodes.
3. Each processing node or workstation cluster at a node is also equipped with a communication coprocessor or a front-end that allows the communication and computation of loads to be carried out simultaneously.
4. The computational load is distributed in a predefined sequence among the available processors.
5. In the adaptive load distribution strategy, the control processor computes the estimated values of z and w for each processing node. Time for this computation time is negligible.
6. The time taken for returning the result of the load processing back to the control processor is small enough to be ignored.

3 Adaptive Load Distribution Strategy

The basic adaptive load distribution strategy consists of two phases. In the first phase (*probe phase*), a small part of the load (*probe installment*) is partitioned and communicated to individual processing nodes. Each node sends back a *communication task completion* (CTC) message when it finishes receiving its load fraction. Thereafter, it sends back a *processing task completion* (PTC) message when it finishes processing this load. Based on these task completion messages, the average bandwidth and average processing capacity of the processing node are estimated. From these, z and w can be computed and (1) can be used to compute the optimal load fractions to be distributed among the individual processor nodes. Then, the second phase, called

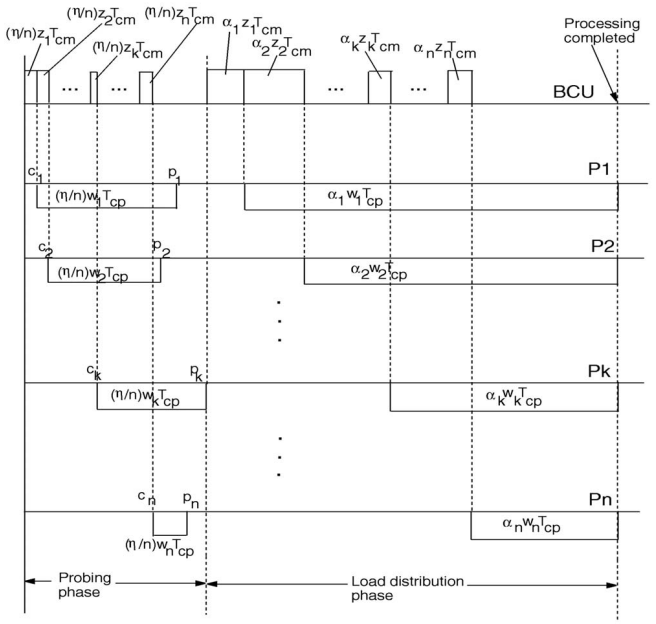


Fig. 4. Timing diagram for the PDD strategy.

the *optimal load distribution phase*, commences. This strategy was first introduced in [18] under the name of feedback strategy for divisible load allocation. However, no detailed analysis was carried out there. The notion of probing to estimate network or processor capability has been used earlier in the multimedia literature for ensuring QoS guarantees [19], [20]. Below, we present three strategies based upon the above basic philosophy. These are not three "different" strategies, but rather, they are increasing refinements of the same basic strategy. The refinements are based on certain lacunae in the strategies, which are illustrated through numerical examples.

3.1 Probing and Delayed Distribution (PDD) Strategy

Probing and delayed distribution (PDD) is the simplest strategy in which the probe installment, which is a small fraction (say η, with $0 < \eta < 1$) of the total load L, is first distributed in equal proportion to the n workstations. So, each workstation is allocated a load of $L_i^p = (\eta/n)L$ during the probing phase. When both the CTC and PTC messages are received from all the processors, the z and w vectors are computed. Then, the remaining load $(1-\eta)L$ is distributed to the processors after partitioning it according to the optimal load distribution equations (1). This process is shown in Fig. 4, where the blocks above the horizontal time line represents the communication times, and the ones below the time line represents the processing time at the nodes. The load is ηL during the probing phase and $(1-\eta)L$ during the load distribution phase. In the figure, the terms L during the probing phase and $(1-\eta)L$ during the load distribution phase have been suppressed for clarity. Note that an optimal distribution of load fractions during the load distribution phase using (1) would ensure that all the computers would stop computing at the same time instant [4].

Let c_i and p_i denote the instants when the CTC (communication task completion) and PTC (processing task completion) messages are received from the ith processor during the probing phase. Let $\hat{z}_i$ and $\hat{w}_i$ be the estimated values of the communication and computation parameters associated with the ith workstation. Then,

$$\hat{z}_i = \frac{c_i - c_{i-1}}{(\eta L/n)T_{cm}}, \quad \hat{w}_i = \frac{p_i - c_i}{(\eta L/n)T_{cp}}, \quad c_0 = 0, \quad i = 1, \ldots n. \tag{2}$$

Let the kth processor be the one that sends its PTC last. Then,

$$k = \arg\max_i \{p_i\}. \tag{3}$$

Below, we propose two algorithms for optimal distribution of the remaining load. The first is based on a closed-form solution and the second on a linear programming formulation.

3.1.1 Closed-Form Solution (CFS) Based Algorithm for PDD

Using the estimated values of the network parameters, the remaining load of $(1-\eta)L$ is distributed starting from time instant p_k. Let the vector of load fractions be $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$, where α_i is the fraction allocated to workstation i. Thus, the load sent to workstation i during the load distribution phase is $L_i^l = (1-\eta)L\alpha_i$. Using the optimality condition that all processors stop computing at the same time instant [4], we obtain,

$$\alpha_i(1-\eta)L\hat{w}_i T_{cp} = \alpha_{i+1}(1-\eta)L\hat{z}_{i+1}T_{cm} + \alpha_{i+1}(1-\eta) L\hat{w}_{i+1}T_{cp}, \quad i = 1, \ldots, n-1 \tag{4}$$

$$\sum_{i=1}^{n} \alpha_i = 1. \tag{5}$$

Solving these equations and substituting from (2), we obtain the following:

$$\alpha_n = \frac{1}{1 + \sum_{l=2}^{n} \prod_{j=l}^{n} \left(\frac{p_j - c_{j-1}}{p_{j-1} - c_{j-1}}\right)}, \quad \alpha_i = \frac{\prod_{j=i+1}^{n} \left(\frac{p_j - c_{j-1}}{p_{j-1} - c_{j-1}}\right)}{1 + \sum_{l=2}^{n} \prod_{j=l}^{n} \left(\frac{p_j - c_{j-1}}{p_{j-1} - c_{j-1}}\right)}, \quad i = 1, \ldots, n-1. \tag{6}$$

So, the optimal load fractions during the load distribution phase may be computed as function of the time instants at which the PTC and the CTC messages were received by the BCU during the probing phase. The total load allocated to workstation i for processing is

$$L_i = L_i^p + L_i^l, \tag{7}$$

where L_i^p and L_i^l are the loads allocated during the probing and the load distribution phases.

$$\begin{aligned} L_i &= \eta\left(\frac{1}{n}\right)L + (1-\eta)L\alpha_i \\ &= \eta\left(\frac{1}{n}\right)L + (1-\eta)L\left[\frac{\prod_{j=i+1}^{n} \left(\frac{p_j - c_{j-1}}{p_{j-1} - c_{j-1}}\right)}{1 + \sum_{l=2}^{n} \prod_{j=l}^{n} \left(\frac{p_j - c_{j-1}}{p_{j-1} - c_{j-1}}\right)}\right]. \end{aligned} \tag{8}$$

The processing time for the workstation cluster is the time at which all the workstations stop computing. Considering the first workstation P_1, we get the processing time as,

$$T = p_k + \alpha_1(1-\eta)L(\hat{z}_1 T_{cm} + \hat{w}_1 T_{cp})$$
$$= p_k + \left\{\frac{p_1 n(1-\eta)}{\eta}\right\}\left[\frac{\prod_{j=2}^{n}\left(\frac{p_j - c_{j-1}}{p_{j-1}-c_{j-1}}\right)}{1+\sum_{l=2}^{n}\prod_{j=l}^{n}\left(\frac{p_j-c_{j-1}}{p_{j-1}-c_{j-1}}\right)}\right]. \quad (9)$$

It has been shown in [4] that the above closed-form solution has to account for "slow" workstations that may have to be "dropped" as they hold up the processing of the whole network. The conditions that identify these workstations are fairly complicated [4, chapter 5, (5.29)]. The above closed-form solution gives optimal load distribution provided that none of the workstations are slow enough to be dropped. However, we would rather look for optimality in the general sense. Thus, the "dropping" condition has to be included right after $\hat{z}$ and $\hat{w}$ are computed in (2). The complete algorithm is then as follows:

Begin {CFS-PDD}
Step 0: *The CTCs $(c_1, \ldots, c_n)$ and PTCs $(p_1, \ldots, p_n)$ of all the workstations are available and the remaining load to be distributed is $(1-\eta)L$.*
Step 1: *Use the load distribution condition, as given in [4, chapter 5, (5.29)], to identify workstations that need to be "dropped."*
Step 2: *Use (6) to compute the normalized load fractions during the load distribution phase, (8) to compute the loads processed by each individual workstation, and (9) to compute the time of completing the computations.*
End {CFS-PDD}

3.1.2 Linear Programming (LP) Based Algorithm for PDD

The LP formulation automatically achieves optimal load distribution with slightly higher computational burden. Also, the algorithm is simpler to use than the closed form solution when there are slow workstations in the network. The LP problem is

$$\min T$$
subject to,
$$p_k + \sum_{j=1}^{i-1}\alpha_j(1-\eta)L\hat{z}_j T_{cm} + \alpha_i(1-\eta)L(\hat{z}_i T_{cm} + \hat{w}_i T_{cp})$$
$$- T \le 0, \quad i = 1, \ldots, n$$
$$\alpha_1 + \alpha_2 + \cdots + \alpha_n = 1, \quad \alpha_1 \ge 0,\ \alpha_2 \ge 0,\ \cdots,\ \alpha_n \ge 0,$$

where the decision variables are $\alpha_1, \ldots, \alpha_n$ and T. On substituting (2), we get

$$\min T$$
subject to,
$$p_k + \sum_{j=1}^{i-1}\left(\frac{c_j - c_{j-1}}{\eta/n}\right)(1-\eta)\alpha_j + \left(\frac{p_i - c_{i-1}}{\eta/n}\right)(1-\eta)$$
$$\alpha_i - T \le 0, \quad i = 1, \ldots, n, \text{ and } c_0 = 0$$
$$\sum_{i=1}^{n}\alpha_i = 1, \quad \alpha_i \ge 0, \quad i = 1, \ldots, n.$$

The solution to this problem yields the optimal load fractions as $\alpha_i(1-\eta L)$, with T as the time of final completion of processing. The complete algorithm is given below:

Begin {LP-PDD}
Step 0: *The CTCs $(c_1, \ldots, c_n)$ and PTCs $(p_1, \ldots, p_n)$ of all the workstations are available and the remaining load to be distributed is $(1-\eta)L$.*
Step 1: *Solve the LP problem given above to obtain the normalized load fractions α_is during the load distribution phase and the time of completion of processing T.*
Step 2: *Use (8) to compute the loads processed by each individual workstation.*
End {LP-PDD}

3.1.3 Discussions

Even in a bus-based system, the exact communication parameter will never be the same for different workstations. Therefore, it is logical to consider the general case of heterogeneous communication speeds even when the underlying communication network is bus-based. The algorithm is adaptive as it does not depend on any assumption on the network resource capabilities in terms of the communication and computation parameters, but still distributes the load with very little sacrifice in optimality. This sacrifice in optimality depends on the size of η. As $\eta \to 0$, we get the same performance as the full information algorithms.

Although simple, the strategy has several drawbacks. Suppose one of the workstation-link pairs is slow, then the time to receive back CTC and PTC messages from this workstation will be very high and distribution of the remaining load will be delayed. We may overcome this by making η very small so that the wait time for receiving the PTC will also be small. Unfortunately, although this appears to be a reasonable solution, it has drawbacks from a practical viewpoint. Usually, image processing or other computational loads require some minimum amount of data for their processing to be meaningful. Moreover, if there is some fluctuation in the network speeds (due to other processes), the averaging effect will not be meaningful unless η is reasonably large. Further, the PDD strategy does not have fault tolerance capability when one (or more) workstations get isolated due to some fault. In such cases, the PTC message will not arrive and the processing of the remaining load will never commence. Of course, one could incorporate an unstructured time-bound abort feature into the algorithm, but we will propose a better solution to this problem in the next algorithm. Finally, we comment that the performance can be further improved by sequencing the load distribution using results from [4]. But, we do not address this issue here.

Example 1. Consider a network consisting of a control processor CP and four workstations $P_1, \ldots, P_4$. We assume $T_{cm} = 1$ and $T_{cp} = 2$. The probe installment is assumed to be 10 percent of the total load $L = 20$. Each workstation gets 0.5 units of the load during the probing phase. So, $\eta = 0.1$ and $L_i^p = 0.5$ for each workstation. The remaining load, which is 18, is distributed among the workstations in an optimal manner after the network speed parameters have been estimated. The results are given in Table 1. The αs in the table show normalized load fractions distributed during the load distribution phase. The L_i^l column shows the load distributions to the workstation during the load distribution phase, and L_i

TABLE 1
Estimated Parameters and Optimal Load Distribution: Example 1

Proc (P_i)	CTC (c_i)	PTC (p_i)	$\hat{z}_i$	$\hat{w}_i$	α_i	L_i^l	L_i	T_i
1	0.05	2.05	0.1	2.0	0.3494	6.2892	6.7892	30.986
2	0.20	5.20	0.3	5.0	0.1357	2.4426	2.9426	30.986
3	0.40	3.40	0.4	3.0	0.2120	3.8160	4.3160	30.986
4	0.50	2.50	0.2	2.0	0.3029	5.4522	5.9522	30.986

gives the total load processed by the ith workstation, including the probing installment.

The results, obtained from both the algorithms (CFS-PDD and LP-PDD) are the same. The LP-PDD was solved using LINDO (Linear Interactive and Discrete Optimizer) [21]. For the sake of completeness, we give the equations for the LP problem below:

$$\min T$$

$$\text{Subject to, } \quad 5.2 + 73.8\alpha_1 - T \leq 0; \quad 5.2 + 1.8\alpha_1 + 185.4\alpha_2 - T \leq 0$$

$$5.2 + 1.8\alpha_1 + 5.4\alpha_2 + 115.2\alpha_3 - T \leq 0; \quad 5.2 + 1.8\alpha_1 + 5.4\alpha_2 + 7.2\alpha_3 + 75.6\alpha_4 - T \leq 0$$

$$\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 = 1; \quad \alpha_1 \geq 0; \quad \alpha_2 \geq 0; \quad \alpha_3 \geq 0; \quad \alpha_4 \geq 0.$$

Let us now compare these results with the perfect information case where the speed parameters are already known, so that the probing phase is not necessary. Using the standard load distribution equations [4] for tree networks and assuming the z and w values to be the same as the values estimated here, the results obtained are shown in Table 2. Thus, the adaptive strategy takes about 8 percent more time than the perfect information case to process the complete load. However, the advantages lie in the fact that the technique is applicable to any network of workstations where the resource capabilities are unknown. Moreover, this gap between the performances of the perfect information case and the unknown information case can be narrowed by making η small. Note that in this example, η has been made 10 percent of the load, but generally, so long as the probe installment has a reasonable size to estimate speed parameters, the technique will work even with smaller probing load sizes. So, for large loads (that is, large values of L), η can be very small indeed.

Example 2. We consider the same network but with the probing load fraction $\eta = 0.05$ (See Table 3).

The probing phase is over in 2.6 time units and the rest of the time is devoted to optimal load distribution and processing. There is only a small improvement (about 0.5 percent) in the total processing time. This example illustrates that reducing the probe load size will not have a great effect on the time performance of the distributed system.

TABLE 2
Optimal Load Distribution for the Perfect Information Case

α_1 (L_1)	α_2 (L_2)	α_3 (L_3)	α_4 (L_4)	T
0.3494 (6.988)	0.1357 (2.714)	0.2120 (4.240)	0.3029 (6.058)	28.6508

TABLE 3
Estimated Parameters and Optimal Load Distribution: Example 2

Proc (P_i)	CTC (c_i)	PTC (p_i)	$\hat{z}_i$	$\hat{w}_i$	α_i	L_i^l	L_i	T_i
1	0.025	1.025	0.1	2.0	0.3494	6.6386	6.8886	30.843
2	0.10	2.60	0.3	5.0	0.1357	2.5783	2.8283	30.843
3	0.20	1.70	0.4	3.0	0.2120	4.0280	4.2780	30.843
4	0.25	1.25	0.2	2.0	0.3029	5.7551	6.0051	30.843

3.2 Probing and Continuous Distribution (PCD) Strategy

The total load is divided into p equal installments of fraction η each, so that $p\eta = 1$. The first installment is further divided into n equal load fractions of size $\eta L/n$ and communicated as the probe installment. Until the last PTC message for the probe installment is received, the subsequent installments are also partitioned and distributed in the same way. This process is shown in Fig. 5, where the last PTC arrives at time p_k. As soon as the last PTC message is received, the communication and computation parameters are computed using (2) and the optimal partitioning of the remaining load (which is $(1 - i\eta)L$, if the ith probing installment is being distributed when the last PTC message was received) is computed. This process of continuously distributing equal installments stops after the last PTC is received. If the jth installment is being distributed when the last PTC is received, then we complete the distribution of this installment before commencing the load distribution phase. This means that the workstations are now ready to receive the optimal load distributions from a time $t_l \geq p_k$, at which point the communication of the last equal installment distribution ceases. We can refine this process further by assuming that t_l is the time at which the current installment

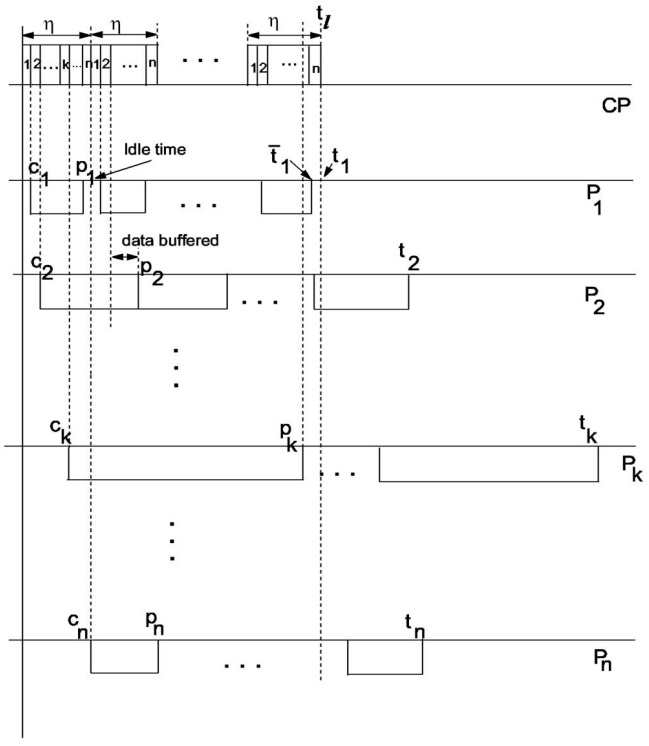


Fig. 5. Timing diagram for the probing phase of PCD strategy.

(to one of the workstations) ceases, but we will not do this here since the benefits are hardly going to be substantial.

Since the equal installment distribution is not optimal, we may have some workstations (usually fast ones) remaining idle during the probe phase. For example, see Fig. 5, which shows workstation P_1 remaining idle for short periods after each installment. Another possible situation is when a workstation (usually a slow one) may receive its installment before it has completed its processing of the previous installment. This would require the latter installment to be buffered at the workstation. This is seen for workstation P_2 in Fig. 5.

After the last PTC is received, all the workstations are ready to receive their installments for the remaining load from time t_l. However, they may not be free to start processing their loads. They would still be busy computing the equal installments until times $\bar{t}_1, \bar{t}_2, \ldots, \bar{t}_n$. However, $\bar{t}_i$ need not be the release times. For a fast workstation (such as P_1 in the figure), it may mean that the workstation will be ready to start computing the optimally distributed load at a time $\bar{t}_i$ much before t_l. In which case, its release time is

$$t_i = \max\{\bar{t}_i, t_l\}. \tag{10}$$

This is true for P_1 for which the release time is $t_1 = t_l$, even though it becomes free at $\bar{t}_1$, while for others $t_i = \bar{t}_i$ in Fig. 5.

Very few algorithms exist in the literature ([5], [8], [12]) for the arbitrary release times case. However, these algorithms are for bus networks, where the communication speed parameter z is assumed to be the same for all links and, hence, are not general enough. Also, even for bus-based systems, the algorithms proposed are applicable to only specific cases which allow multi-installment distributions and change in the sequence of distributions. Below, we propose two algorithms, the first based on closed-form solution (CFS) and the second on linear programming (LP), for which the sequence of load distribution is fixed and load is distributed in a single installment to each workstation. We do not attempt a proof of optimality of the first (CFS) algorithm (although we conjecture that the algorithm is indeed optimal) since the proof is likely to be fairly elaborate and is beyond the scope of this paper. However, the LP-based algorithm, by definition, will give optimal results.

3.2.1 Closed-Form Solution (CFS) Based Algorithm for PCD

Let t_l be the time instant from which the remaining load $(1 - q\eta)L$ is to be distributed. The release times of the workstations are $t_1, t_2, \ldots, t_n$, with $t_i > t_l$ for all i. Let us obtain the load distribution for the set of workstations under consideration (denoted as I, where I initially contains all the workstations), ignoring the release time constraints. Then, the load distribution is given by

$$L_i^l = \alpha_i(1 - q\eta)L, \text{ for all } i \in I, \tag{11}$$

where α_i is as given in (6). This is the load distribution that we would get if the workstations were idle from the beginning or the communication of load to the workstations get completed only after the workstation's release time. Now, if

$$t_i \le t_l + \sum_{j=1}^{i} L_j^l \hat{z}_j T_{cm} \tag{12}$$

for all $i = 1, \ldots, n$, then (11) is the required load distribution and we need look no further. Note that the RHS of (12) is just the time at which P_i begins its computations. Otherwise, if (12) is violated for some i, then we partition the workstation sets into I_1 and I_2, where I_1 contains workstation indices that satisfy (12) and I_2 contains those which do not. Thus, members of I_1 are not affected by ignoring the release time constraints, but members of I_2 will violate the release time constraints if the load distribution is computed from (6) and (11). Given this classification, we may rewrite the load distribution equations as

$$T - t_l = \sum_{j=1}^{i} \alpha_j(1 - q\eta)L\hat{z}_j T_{cm} + \alpha_i(1 - q\eta)L\hat{w}_i T_{cp}, \quad \text{for } i \in I_1 \tag{13}$$

$$T = t_i + \alpha_i(1 - q\eta)L\hat{w}_i T_{cp}, \text{ for } i \in I_2, \tag{14}$$

where, at time T, all workstations stop computing their load fractions. These, coupled with the normalizing equation $\sum_{j=1}^{n} \alpha_j = 1$, give $n + 1$ equations in $n + 1$ unknowns. Let

$$\begin{aligned} \zeta_i &= (1 - q\eta)L\hat{z}_i T_{cm} = n(1 - q\eta)(c_i - c_{i-1})/\eta \\ \omega_i &= (1 - q\eta)L\hat{w}_i T_{cp} = n(1 - q\eta)(p_i - c_i)/\eta \\ \zeta_i + \omega_i &= (1 - q\eta)L(\hat{z}_i T_{cm} + \hat{w}_i T_{cp}) = n(1 - q\eta)(p_i - c_{i-1})/\eta, \\ & \quad i = 1, 2, \ldots, n. \end{aligned} \tag{15}$$

Now, define two $(n + 1) \times (n + 1)$ matrices A_1 and A_2 for the workstation sets I_1 and I_2 and two $(n + 1) \times 1$ column matrices b_1 and b_2, as follows:

$$A_1 = \begin{bmatrix} 1 & -(\zeta_1+\omega_1) & 0 & 0 & \cdots & 0 \\ 1 & -\zeta_1 & -(\zeta_2+\omega_2) & 0 & \cdots & 0 \\ 1 & -\zeta_1 & -\zeta_2 & -(\zeta_3+\omega_3) & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & -\zeta_1 & -\zeta_2 & -\zeta_3 & \cdots & -(\zeta_n+\omega_n) \\ 0 & 1 & 1 & 1 & \cdots & 1 \end{bmatrix} \tag{16}$$

$$A_2 = \begin{bmatrix} 1 & -\omega_1 & 0 & 0 & \cdots & 0 \\ 1 & 0 & -\omega_2 & 0 & \cdots & 0 \\ 1 & 0 & 0 & -\omega_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & 0 & 0 & \cdots & -\omega_n \\ 0 & 1 & 1 & 1 & \cdots & 1 \end{bmatrix}; \; b_1 = \begin{bmatrix} t_l \\ t_l \\ t_l \\ \vdots \\ t_l \\ 1 \end{bmatrix}; \; b_2 = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ \vdots \\ t_n \\ 1 \end{bmatrix}. \tag{17}$$

Then, (13) and (14) may be expressed in the matrix form as

$$Ax = b, \tag{18}$$

where $x = [T \;\; \alpha_1 \;\; \alpha_2 \;\; \cdots \;\; \alpha_n]'$ and, for $i = 1, \ldots, n$, the ith row of A is the ith row of A_1 if $i \in I_1$ and the ith row of A_2 if $i \in I_2$. Similarly, for $i = 1, \ldots, n$, the ith element of the column vector b is the ith element of b_1 if $i \in I_1$ and the ith element of b_2 if $i \in I_2$. The $(n + 1)$th row of A and the corresponding element of b are the same as those of either A_1 or A_2 and b_1 or b_2, respectively. Solving (18) yields the values of the load fractions α_is and the total processing time T. With these values, is it possible that some of the workstations in I_1 will now violate (12)? The answer is "no" since these workstations will now process larger load fractions than they were earlier called upon to process,

which implies that they will start their computations *later* than in the earlier case. Since t_i depends only on the equal installment distributions, (12) will not be violated.

If all the α_is so obtained are nonnegative, then we have an optimal load distribution. Otherwise, suppose some of the α_is are negative, then these workstations draw "negative" load, implying that they have release times that are so high that the other workstations will be able to finish computing their loads before these "negative" workstations even start computing theirs. We can "drop" these "negative" workstations by assigning them zero loads and reformulate the problem for the set of remaining workstations. By continuing this iterative process, we will ultimately converge to a situation when all the workstations that participate in the load distribution are assigned nonnegative loads when (18) is solved. This process converges because at none of the steps will all the workstations get dropped. To prove the optimality of this procedure, we need to ensure that while dropping workstations we do not drop one that should actually contribute in the load processing. In other words, if we have two or more "negative" workstations, then by dropping only a few of them, we may be able to retain the positivity of the others and, thus, make them participate in the load distribution process. We will not analyze this situation here. However, note that by dropping all such "negative" workstations we can converge to a feasible load distribution policy. Another point that needs to be explained is that the load distribution condition that were derived in [4] and used in the CFS-PDD algorithm is no longer valid here since the workstations start at different release times.

Begin {CFS-PCD}
Step 0: *The CTCs* $(c_1, \ldots, c_n)$ *and PTCs* $(p_1, \ldots, p_n)$ *of all the workstations are available. The remaining load to be distributed is* $(1-\eta)L$. *The release times of the workstations are* $t_1, t_2, \ldots, t_n$, *with* $t_i > t_l$ *for all* i.
Step 1: *Define the set* I *of workstation indices and solve (11) using (6).*
Step 2: *Evaluate (12). If these inequalities are satisfied for all workstations, then STOP. Else, define sets* I_1 *and* I_2.
Step 3: *Formulate (18) and solve. If all the* α_is *are positive, then STOP.*
Step 4: *Drop the "negative" processors, update the set* I, *and go back to Step 1.*
End {CFS-PCD}

3.2.2 Linear Programming (LP) Based Algorithm for PCD

The LP formulation is as follows:

$$\begin{aligned}
&\min T \\
&\text{subject to,} \\
&t_l + \sum_{j=1}^{i-1} \alpha_i (1-q\eta) L \hat{z}_i T_{cm} + \alpha_i (1-q\eta) L (\hat{z}_i T_{cm} + \hat{w}_i T_{cp}) \\
&\quad - T \le 0 \\
&t_i + \alpha_i (1-q\eta) L \hat{w}_i T_{cp} - T \le 0, \quad i = 1, \ldots, n \\
&\alpha_1 + \alpha_2 + \cdots + \alpha_n = 1, \quad \alpha_1 \ge 0, \quad \alpha_2 \ge 0, \quad \cdots, \quad \alpha_n \ge 0.
\end{aligned}$$

On substituting (2), the LP problem becomes

$$\begin{aligned}
&\min T \\
&\text{subject to,} \\
&t_l + \sum_{j=1}^{i-1} \left(\frac{c_j - c_{j-1}}{\eta/n} \right) (1-\eta)\alpha_j + \left(\frac{p_i - c_{i-1}}{\eta/n} \right) (1-q\eta)\alpha_i \le T, \\
&i = 1, \ldots, n, \text{ and } c_0 = 0 \\
&t_i + \left(\frac{p_i - c_i}{\eta/n} \right) (1-q\eta)\alpha_i \le T, \quad i = 1, \ldots, n \\
&\sum_{i=1}^{n} \alpha_i = 1, \quad \alpha_i \ge 0, \quad i = 1, \ldots, n.
\end{aligned}$$

The complete algorithm is given below:

Begin {LP-PCD}
Step 0: *The CTCs* $(c_1, \ldots, c_n)$ *and PTCs* $(p_1, \ldots, p_n)$ *of all the workstations are available. The remaining load to be distributed is* $(1-\eta)L$. *The release times of the workstations are* $t_1, t_2, \ldots, t_n$, *with* $t_i > t_l$ *for all* i.
Step 1: *Solve the LP problem given above to obtain the normalized load fractions* α_is *during the load distribution phase and the time of completion of processing* T.
Step 2: *Compute the loads processed by each individual workstation.*
End {LP-PCD}

3.2.3 Discussions

Although the PCD strategy avoids the possibility that fast workstation link pairs remain idle for a long time while the BCU is waiting for the last PTC to enable it to compute network parameters, it still has to deal with the situation that, if a workstation is very slow, then a substantial part of the load would have been distributed in a nonoptimal manner before sufficient data is available for computing the network parameters. In fact, the whole of the load may have been distributed by the time this happens, and then the slow workstation-link pair will decide the time performance, which will be far from the optimal distribution. In fact, this may make PCD slower than PDD. We will see an instance of this in Example 3 below. Hence, we need to develop a technique by which the slow workstation-link pairs do not get a large fraction of the load and, consequently, do not slow down the processing of the load. We will address this problem next. As in PDD, it may be possible to improve performance by changing the sequence of load distribution. However, results available in the literature, for the case when processors are released at arbitrary times, is mainly of a heuristic nature since an exact solution can be achieved only through combinatorial optimization algorithms of high complexity.

Example 3. The network is the same as before, and $\eta = 0.1$ as in Example 1 (for PDD). Since $p_k = p_2 = 5.2$ and $c_4 = 0.5$, the distribution of η fraction of the total load takes just 0.5 units of time and this load, when distributed in equal installments to all the workstations, gets completely processed in 5.2 units of time. So, according to the PCD strategy, until the time 5.2 units, the control processor continues to distribute the load in

equal installments of η (which is again partitioned equally among the workstations). But, the total load will then be distributed in just 10 such installments taking $10 \times 0.5 = 5$ units of time. So, the total load is completely distributed in equal and nonoptimal installments before the last PTC is received and there is no load left to be distributed optimally! This demonstrates the perils of selecting too large an η in the PCD strategy. The finish times of the workstations then is easily computed from

$$T_i = c_i + 10(p_i - c_i), \tag{19}$$

yielding $T_1 = 20.05,\ T_2 = 50.20,\ T_3 = 30.40,\ T_4 = 20.50$. Hence, the time at which the processing of the complete load is over is, $T = T_2 = 50.20$ time units.

Example 4. Select the probe fraction as $\eta = 0.05$, as in Example 2. The corresponding CTCs and PTCs are as given in Table 3, and so are the estimated network parameters. Now, we can use either the CFS or the LP algorithm to solve this problem. We omit the step-by-step details of the solution process which is available in the report [5]. In the CFS algorithm, the steps have to loop twice through the iterations. The optimal load distribution during the load distribution phase is $\alpha_1 = 0.41675,\ \alpha_2 = 0.0,\ \alpha_3 = 0.17275,\ \alpha_4 = 0.41050$. Including the probe fraction and the continuous distribution fractions, the final optimal load allocations to each of the workstations is $L_1 = 6.50075,\ L_2 = 2.75,\ L_3 = 4.30475,\ L_4 = 6.4445$, and the time for completing the processing is $T = 27.6$. Note that the entire load processed by P_2 is the load that it gets during the equal installment distribution phase. In fact, it finishes computing this load only at time 27.6 which is greater than the time 26.0681 at which the other workstations finish computing their loads. Thus, the processing time of the PCD strategy is 27.6 time units.

The insights that we gain from this example are: 1) The optimum time of 27.6 is better than the perfect information case where the optimum time was 28.6508. This may appear counter-intuitive, but is easily explained. The perfect information strategy assumed that the load is distributed in a single installment to each workstation. Whereas, in the PCD strategy, the equal installment distribution is done in several (in this example, 11) installment of ηL each. This gives the standard advantage that multi-installment strategy has over single installment strategies of allowing a workstation to start computing without waiting for the whole load to be received [4]. 2) This example also shows that the load distribution conditions proposed in [4] are not valid here because according to those conditions, which depend only on the values of z_i and w_i and not on the release times t_i, all workstations are eligible to participate in the processing of the load. But, this example shows that this is not the case and, so, a general load distribution condition must also take into account the processor release times. In any case, our algorithms give a simple numerical approach to identify workstations that need to be eliminated from the load distribution process.

3.3 Probing and Selective Distribution (PSD) Strategy

The basic solution methodology of PSD strategy is the same as that of PCD except for the modification that optimal partitioning of some fraction of the total load is done only among a subset of the workstations (actually, those workstations that have sent back their PTCs). Below, we describe this strategy in which the first two steps are the same as in PCD:

1. For the probing phase, consider the first ηL fraction of the load. Partition it into n equal portions and distribute in the sequence of ascending order of workstation index.
2. Continue doing this equal installment load distribution during the probing phase until the first PTC is received at the control unit, say, from workstation P_i.
3. The next installments, each of size ηL, are now sent only to P_i until another PTC is received, say, from P_j.
4. The next two installments, together of size $(2\eta L)$, are now partitioned *optimally* among workstations P_i and P_j, following the same sequence of ascending workstation indices, using the PCD algorithm (either CFS or LP based), and with t_l as the time when the current installment (of size ηL) finishes getting communicated to P_j, and the release times, t_i and t_j, are computed using the estimated network parameters for these two processing nodes and the loads already distributed to them.
5. Continue distributing $2\eta L$ sizes of loads to P_i and P_j until another PTC, say, from workstation P_k is received. Then, the set of workstations expands to three and the PCD algorithm is used to distribute $3\eta L$ loads optimally between these three nodes.
6. Continuing in this way, the workstation set keeps on expanding as PTCs are received and the load (which is an integer multiple of the basic installment size of ηL, the integer being equal to the number of nodes in the workstation set) gets distributed optimally among the members of the workstation set. The load distribution at each stage is obtained by using the PCD algorithm.
7. This process continues until all the PTCs are received and the workstation set contains all the workstations. At this point, there are two options:
   a. Continue in the same way by taking sizes of $n(\eta L)$ load and distributing them optimally, until all the load is exhausted.
   b. Take all the remaining load and partition it optimally among all the workstations.

Some points to remember are: 1) At every stage, take $\min\{q\eta L,\ L_r\}$ as the load to be distributed, where q is the number of workstations in the set, and L_r is the remaining load to be processed. 2) Each optimal distribution of $q\eta L$ requires fresh computation of t_l and the release times for each workstation in the workstation set. 3) The advantage of this algorithm is that a very slow link-workstation pair will get only a small fraction of the total load, and the one which has a fault will not get any load (except the first probe

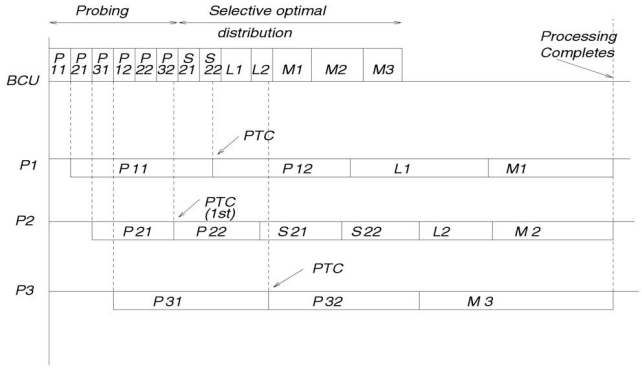


Fig. 6. Probing and selective distribution strategy.

installment of $\eta L/n$). Thus, these workstations are less likely to hold up the computations.

Fig. 6 shows a possible distribution of load among three workstations using the PSD strategy. Here, P_{11}, P_{21}, P_{31}, each of size $\eta L/3$, are the probing installments. Since no PTCs are received, the load distribution continues with P_{12}, P_{22}, P_{32}, again, each of size $\eta L/3$. Just before P_{32} finishes getting communicated, the first PTC from P_2 is received. Then, $S_{21} = S_{22} = \eta L$ are communicated to P_2 only (the figure is not to scale). After the second PTC is received from P_1, the load $L_1 + L_2 = 2\eta L$ is partitioned optimally among P_1 and P_2. Then, the third PTC is received and the remaining load $M_1 + M_2 + M_3$ is partitioned optimally among all the three processors.

Example 5. We consider the same network of workstations. As in Examples 1 and 3, we assume η to be 0.1. As before, we may use either CFS or LP to solve this problem. The details are available in [5]. The time for completely processing the total load is 26.55625, which is the time at which the workstations P_1, P_3, and P_4 stop. But, P_2 gets only the equal installment load and finishes processing it at time 25.2. P_2 does not get any load during the optimal load distribution phase. The final load distribution among processors is $L_1 = 6.6265625,\ L_2 = 2.5,\ L_3 = 4.359379,\ L_4 = 6.5140625$. This shows that even for large η, PSD does not have the type of nonoptimal performance as PCD.

Example 6. Consider the same network, but with the probe fraction $\eta = 0.05$, as assumed in Examples 2 and 4. The final load distribution among processors is $L_1 = 6.552082,\ L_2 = 2.6134,\ L_3 = 4.338886,\ L_4 = 6.495832$. All the workstations participate in all the phases of load distribution. The time performance is also the best with $T = 26.233$. The details are available in [5].

3.4 Comparison

In Table 4, we compare the numerical results obtained for the above network of workstations against the perfect-information single installment case and the perfect-information multi-installment case in the limit (that is, when the number of installments become infinity). In fact, it has been shown in [4] that the latter is equivalent to considering a network with no communication delay. Hence, this is the absolute best performance that can be obtained.

TABLE 4
Comparison of Processing Times

Load Distribution Algorithms	Time for $\eta = 0.1$	Time for $\eta = 0.05$
Single Installment Perfect Information	28.651	28.651
Multi(∞)-installment Perfect Information (Ideal case)	26.087	26.087
PDD	30.986	30.843
PCD	50.200†	26.556
PSD	27.400	26.233

† Nonoptimal solution

4 An Experimental Implementation

All the above numerical examples were for an ideal model of a network of workstations, not subject to any external disturbances, fluctuations, or uncertainties. A real network will have all this, the main cause being various types of other processes that routinely run in a distributed network, appearing sporadically and causing fluctuations in the communication bandwidths and computational capabilities. An example of this can be seen in [7, Fig. 2, p. 66]. To demonstrate the applicability of our algorithms to a real system, we set up a seven node system, with Windows XP OS, in which one node acted as the control processor (CP) node and the rest as processing nodes. The details of the nodes are given in Table 5. We would like to clarify that our objective was not to evaluate the three strategies, but rather to show that the three strategies, from the simplest PDD to the complex PSD, are implementable in a real system and validate them through some reasonable simulation. A systematic evaluation of these strategies should consider the effect of the practical factors, mentioned above, that are not taken into account explicitly in the load distribution, computation, communication, and network models assumed. The effect of these unmodelled factors will be of great interest. This kind of extensive simulation will require development of an elaborate simulation platform that is way beyond the scope of this paper and is a matter of future research effort.

A load consisting of several thousands matrix multiplications involving matrices of dimension 100×100 was considered for the experiment. Such requirements of matrix multiplications are typical in designing industrial microwave ovens and for finite-element computations for large-scale structural engineering applications, where the matrices are even larger. We conducted several experiments of this problem on this platform. But, since the results of these are not markedly different, and due to space limitations, we

TABLE 5
Node Specifications for the Experiment

Node	CPU	Memory
CP	P4 1.6 GHz	512 MB
P_1	P3 500 MHz	128 MB
P_2	P4 1.8 GHz	512 MB
P_3	P3 800 MHz	256 MB
P_4	P3 866 MHz	512 MB
P_5, P_6	P3 800 MHz	512 MB

TABLE 6
Experimental Load and Its Computation

MATRIX LOAD		COMPUTATION	LOAD TYPE
A_1	$\rightarrow$	$(A_1)^M$	
$\vdots$	$\vdots$	$\vdots$	Probing load to $P_1 - P_6$
A_{600}	$\rightarrow$	$(A_{600})^M$	in equal sizes
A_{601}	$\rightarrow$	$(A_{601})^M$	
$\vdots$	$\vdots$	$\vdots$	Remaining load to be
A_L	$\rightarrow$	$(A_L)^M$	distributed optimally

report only one of them. In [5], details of the other simulations are given.

The load and the computations involved are given in Table 6. Each matrix A_i is multiplied M times, and there are a total of L such matrices. The first 100 matrices were sent as probe installment to P_1, the next 100 matrices to P_2, and so on. So, the total number of matrices sent for probing were 600. The remaining load $L_r = L - 600$ is now optimally distributed among the six workstations depending on the estimated network parameters and using the algorithms proposed in the paper. An MPI [6] implementation on the seven node system was done. We assume $M = 300$ and $L_r = 10,000$ so that $\eta = 0.05660$.

For the probing and delayed distribution (PDD) strategy, the results are shown in Table 7. The load distributions are such that the processing nodes stop computing at times that are not exactly equal, but close. The maximum fluctuation over a mean value of 71.6 is about 3.9 percent. This fluctuation is due to the presence of background jobs in the machines. The corresponding results for PCD and PSD are given in Tables 8 and 9. In all the tables, c_i, p_i, and T_i are in seconds.

These results should be interpreted keeping in mind that the system was not dedicated to these experiments and had other users who continuously submitted background jobs to the system (which is a desirable situation while testing the efficacy of the algorithms). Since the experiments were carried out at different times and days, the results were likely to differ. This can also be seen from the values of the estimated network parameters w and z. Although the algorithms were developed for workstation clusters where the presence of communication coprocessors or front-ends were assumed to be available for communication offloading, in the network of PCs used for these experiments, this was not the case. So, we had to factor out this extra communication delay from the recorded time values. We simulated the presence of front-ends through this technique, but it is likely that system load fluctuations and load independent communication latency overheads, which were beyond our control, would have contributed as unmodelled noise. The strength of these algorithms, that emerges from these experiments, is that they can handle fluctuations in the network resource capability and are able to allocate loads in a fairly optimal manner. This makes our technique adaptive and truly robust. These experiments, we believe, will provide a strong support to the divisible load scheduling research results and their applicability to real systems.

TABLE 8
Experimental Results Using PCD: $\eta = 0.05660$

Proc. (P_i)	CTC (c_i)	PTC (p_i)	$\hat{z}_i$	$\hat{w}_i$	α_i	L_i	T_i
1	0.480190	4.553135	0.004802	0.020366	0.1600	1700	67.918
2	0.952413	2.849277	0.004723	0.009485	0.2751	2851	68.275
3	1.412117	3.875651	0.004597	0.012318	0.1785	1885	69.136
4	1.864799	4.134433	0.004527	0.011349	0.1615	1715	69.608
5	2.342813	4.839369	0.004780	0.012484	0.1232	1332	68.240
6	2.834644	5.382188	0.004919	0.012739	0.1017	1117	71.096

5 Conclusions

In this paper, we describe a new concept in load partitioning and distribution of divisible loads in workstation clusters that enables the implementation of conventional load partitioning algorithms existing in the divisible load scheduling literature to situations when exact values for the network bandwidth and processing capability are not available. This expands the horizon of conventional divisible load scheduling considerably and makes its implementation practically feasible. The rich literature on divisible load scheduling that has accrued over the past decade in terms of load distribution strategies under various conditions can now be implemented as a combined network parameter estimation and adaptive scheduling algorithm. Apart from implementation, there are considerable analytical and theoretical challenges that arise while designing these algorithms and analyzing their performance. A point to note is that the word "adaptive" is used to convey the idea that this approach is implementable in systems that have fluctuating network parameter values. However, in this approach, we create a static model of the network rather than adapt to dynamic changes. But, this approach has the potential to be used in a dynamic situation too in a simple feedback mode. In that sense, our approach is adaptive. A truly adaptive approach will of course need to take into account the dynamically changing network environment and its processing and communication capacity. It also opens up immense possibilities of applying elegant filtering techniques to estimate network parameters

TABLE 7
Experimental Results Using PDD: $\eta = 0.05660$

Proc. (P_i)	CTC (c_i)	PTC (p_i)	$\hat{z}_i$	$\hat{w}_i$	α_i	L_i	T_i
1	0.466584	4.533957	0.004666	0.020338	0.1564	1664	72.546
2	0.915028	2.694977	0.004485	0.008900	0.2855	2955	70.151
3	1.359354	3.806387	0.004444	0.012236	0.1757	1857	69.639
4	1.811722	4.071421	0.004524	0.011299	0.1586	1686	72.423
5	2.262251	4.762126	0.004506	0.012500	0.1214	1314	70.628
6	2.729852	5.236098	0.004676	0.012532	0.1024	1124	74.477

TABLE 9
Experimental Results Using PSD: $\eta = 0.05660$

Proc. (P_i)	CTC (c_i)	PTC (p_i)	$\hat{z}_i$	$\hat{w}_i$	α_i	L_i	T_i
1	0.484788	4.544112	0.004848	0.020298	0.1600	1700	67.906
2	0.963180	2.824774	0.004784	0.0093086	0.2776	2876	68.394
3	1.402900	3.855164	0.004397	0.012262	0.1787	1887	69.150
4	1.868786	4.130869	0.004659	0.011311	0.1606	1706	68.459
5	2.346037	4.874032	0.004773	0.012641	0.1209	1309	71.340
6	2.818742	5.340477	0.004727	0.012700	0.1022	1122	73.374

given some statistics on the variation of these parameters and the past measurements, making the approach truly adaptive. Use of multi-installment strategies and affine models of load distribution and processing are other practical refinements. Research in these directions is presently under progress.

ACKNOWLEDGMENTS

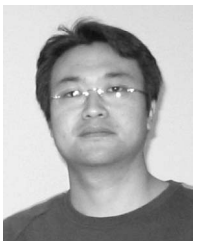
This work was in part supported by the Media Service Research Center (MSRC-ITRC), Ministry of Information and Communication, Korea. The authors would like to thank T.G. Robertazzi for drawing their attention to the use of probing technique in the QoS literature and the reviewers for their excellent comments and suggestions.

REFERENCES

[1] Y.C. Cheng and T.G. Robertazzi, "Distributed Computation with Communication Delays," *IEEE Trans. Aerospace and Electronic Systems,* vol. 24, pp. 700-712, 1988.
[2] Y.C. Cheng and T.G. Robertazzi, "Distributed Computation for a Tree Network with Communication Delays," *IEEE Trans. Aerospace and Electronic Systems,* vol. 26, no. 3, pp. 511-516, 1990.
[3] R. Agrawal and H.V. Jagadish, "Partitioning Techniques for Large-Grained Parallelism," *IEEE Trans. Computers,* vol. 37, no. 12, pp. 1627-1634, 1988.
[4] V. Bharadwaj, D. Ghose, V. Mani, and T.G. Robertazzi, *Scheduling Divisible Loads in Parallel and Distributed Systems.* CS Press, 1996.
[5] M. Drozdowski, "Selected Problems of Scheduling Tasks in Multiprocessor Computer Systems," Poznan Univ. of Technology Press Series Monographs: No. 321, Poznan, Poland, 1997.
[6] V. Bharadwaj, D. Ghose, and T.G. Robertazzi, "Divisible Load Theory: A New Paradigm for Load Scheduling in Distributed Systems," *Cluster Computing,* vol. 6, no. 1, pp. 7-17, 2003.
[7] T. Robertazzi, "Ten Reasons to Use Divisible Load Theory," *Computer,* vol. 36, no. 5, pp. 63-68, 2003.
[8] V. Bharadwaj, H.F. Li, and T. Radhakrishnan, "Scheduling Divisible Loads in Bus Networks with Arbitrary Processor Release Times," *Computer Math. Applications,* vol. 32, no. 7, pp. 57-77, 1996.
[9] J. Sohn and T.G. Robertazzi, "Optimal Time-Varying Load Sharing for Divisible Loads," *IEEE Trans. Aerospace and Electronic Systems,* vol. 34, pp. 907-924, 1998.
[10] V. Bharadwaj and G. Barlas, "Efficient Scheduling Strategies for Processing Multiple Divisible Loads on Bus Networks," *J. Parallel and Distributed Computing,* vol. 62, no. 1, pp. 132-151, 2002.
[11] K. Li, "Parallel Processing of Divisible Loads on Partitionable Static Interconnection Networks," *Cluster Computing,* vol. 6, no. 1, pp. 47-55, 2003.
[12] V. Bharadwaj and G. Barlas, "Scheduling Divisible Loads with Processor Release Times and Finite Size Buffer Capacity Constraints," *Cluster Computing,* vol. 6, pp. 63-74, 2003.
[13] M. Drozdowski and P. Wolniewicz, "Out-of-Core Divisible Load Processing," *IEEE Trans. Parallel and Distributed Systems,* vol. 14, no. 10, pp. 1048-1056, Oct. 2003.
[14] G. 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 and Distributed Systems,* vol. 9, no. 5, pp. 429-441, May 1998.
[15] J. Sohn, T.G. Robertazzi, and S. Luryi, "Optimizing Computing Costs Using Divisible Load Analysis," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 3, pp. 225-234, Mar. 1998.
[16] S. Charcranoon, T.G. Robertazzi, and S. Luryi, "Parallel Processor Configuration Design with Processing/Transmission Costs," *IEEE Trans. Computers,* vol. 40, no. 9, pp. 987-991, Sept. 2000.
[17] S. Bataineh and M. Al-Ibrahim, "Effect of Fault-Tolerance and Communication Delay on Response Time in a Multi-Workstation System with Bus Topology," *Computer Comm.,* vol. 17, pp. 843-851, 1994.
[18] D. Ghose, "A Feedback Strategy for Load Allocation in Workstation Clusters with Unknown Network Resource Capabilities Using the DLT Paradigm," *Proc. Int'l Conf. Parallel and Distributed Processing Techniques and Applications,* vol. 1, pp. 425-428, July 2002.
[19] K. Nahrstedt, H.-H. Chu, and S. Narayan, "QoS Aware Resource Management for Distributed Multimedia Applictions," *J. High Speed Networks,* vol. 7, pp. 229-257, 1998.
[20] B. Li, D. Xu, and K. Nahrstedt, "An Integrated Runtime QoS-Aware Middleware Framework for Distributed Multimedia Applications," *Multimedia Systems,* vol. 8, pp. 420-430, 2002.
[21] http://www.lindo.com, 2005.
[22] D. Ghose et al., "Adaptive Divisible Load Scheduling Strategies for Workstation Clusters with Unknown Network Resources," Technical Report KNU/CI/MSL/001/2003, Multimedia Security Laboratory, Kangwon Nat'l Univ., July 2003.
[23] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface,* second ed., Cambridge, Mass.: MIT Press, 1999.

Debasish Ghose (M '03) received the BSc (Engg) degree from the Regional Engineering College, Rourkela (presently, National Institute of Technology) and the Master of engineering and PhD degree in engineering from the Indian Institute of Science, Bangalore. He has been in the faculty of the Aerospace Engineering Department at IISc from 1990 and is presently a professor there. He has held several visiting faculty positions at universities such as the University of California at Los Angeles and the Kangwon National University in South Korea, among others. Load scheduling algorithms and their analysis are among his research interests in the general area of decision making strategies. He is the coauthor of a book entitled *Scheduling Divisible Loads in Parallel and Distributed Systems* (CS Press and John Wiley) and a guest editor for a special issue on scheduling published by the journal *Cluster Computing*. He is a member of the IEEE.

Hyoung Joong Kim (S '86-M '89) received the BS, MS, and PhD degrees from Seoul National University, Seoul, Korea, in 1978, 1986, and 1989, respectively. He joined the faculty of the Department of Control and Instrumentation Engineering, Kangwon National University, Chunchon, Korea, in 1989, where he is currently a professor. He was a visiting scholar at the University of Southern California from 1992-1993. From 1998-2000, he was the prime investigator of the iPCTV Project developing interactive digital television and implemented MHP, ATVEF, and DASE specifications and performed conformance tests. He has been the prime investigator of the iMS (Interactive Media Solution) developing the end-to-end solution for data broadcasting system since 2000. He was the founder of the International Workshop on Digital Watermarking (IWDW) and served as a cochair of the Technical Program Committee of the workshop. He edited *Digital Watermarking*, *Lecture Notes in Computer Science*, vol. 2613, with Dr. Fabien A.P. Petitcolas. He served as a guest editor of the *IEEE Transactions on Circuits and Systems for Video Technology* in 2003. Since 2003, he has been the director of the Media Service Research Center (MSRC-ITRC) sponsored by the Ministry of Information and Communication. His research interests include parallel computing, multimedia computing, and multimedia security. He is a member of the IEEE.



Tae Hoon Kim received the BS and MS degrees from the Department of Control and Instrumentation Engineering at Kangwon National University, Korea, in 2002 and 2004, respectively. His interests are in the area of multimedia systems. He is associated with the Multimedia Systems Laboratory in the same department.