Cluster Computing 7, 191–207, 2004
© 2004 Kluwer Academic Publishers. Manufactured in The Netherlands.

Design and Performance Analysis of Divisible Load Scheduling Strategies on Arbitrary Graphs

JINGNAN YAO * and BHARADWAJ VEERAVALLI
Open Source Software Laboratory (http://opensource.nus.edu.sg), Department of Electrical and Computer Engineering, The National University of Singapore, 4 Engineering Drive 3, Singapore 117576

Abstract. In this paper, we consider the problem of scheduling divisible loads on arbitrary graphs with the objective to minimize the total processing time of the entire load submitted for processing. We consider an arbitrary graph network comprising heterogeneous processors interconnected via heterogeneous links in an arbitrary fashion. The divisible load is assumed to originate at any processor in the network. We transform the problem into a multi-level unbalanced tree network and schedule the divisible load. We design systematic procedures to identify and eliminate any redundant processor–link pairs (those pairs whose consideration in scheduling will penalize the performance) and derive an optimal tree structure to obtain an optimal processing time, for a fixed sequence of load distribution. Since the algorithm thrives to determine an equivalent number of processors (resources) that can be used for processing the entire load, we refer to this approach as *resource-aware optimal load distribution* (RAOLD) algorithm. We extend our study by applying the optimal sequencing theorem proposed for single-level tree networks in the literature for multi-level tree for obtaining an optimal solution. We evaluate the performance for a wide range of arbitrary graphs with varying connectivity probabilities and processor densities. We also study the effect of network scalability and connectivity. We demonstrate the time performance when the point of load origination differs in the network and highlight certain key features that may be useful for algorithm and/or network system designers. We evaluate the time performance with rigorous simulation experiments under different system parameters for the ease of a complete understanding.

Keywords: load distribution, arbitrary graphs, spanning tree networks, divisible loads, processing time, communication delays

1. Introduction

Network-based scheduling and processing of computationally intensive loads has been proven to be an efficient and successful way in the past decade for various applications. Applications such as large scale simulation in computational fluid-dynamics, remote control/sensor systems, solutions to N-body problems, computational biology, etc. [1–4] are few such examples. Processing and handling such a large volume of traffic demands a clever design of polices to partition, communicate, and to process the load on a network of computers. A vast amount of literature exists in the area of load sharing/balancing on networks [5]. These studies attempt to carefully optimize the available computational power in the network by sharing/balancing the loads. The theory of scheduling and processing of divisible loads, referred to as *divisible load theory* (DLT), since its origin in 1988 [6], has stimulated considerable interest among researchers in the field of parallel and distributed systems. In contrast to the indivisible load model, the loads considered for scheduling and processing in DLT are assumed to be very large in size, homogeneous, and are arbitrarily divisible.

The load being assumed to be arbitrarily divisible, each partitioned portion of the load can be independently processed on any processor on the network and each portion demands identical processing requirements. DLT adopts a linear mathematical modeling of the processor speed and communication link speed parameters. In this model, the communication time delay is assumed to be proportional to the amount of load that is transferred over the channel, and the computation time is proportional to the amount of load assigned to the processor. The primary objective in the research of DLT is to determine the *optimal fractions* of the entire load to be assigned to each of the processors such that the total processing time of the entire load is a minimum. Compilation of all the research contributions in DLT until 1996 can be found in the monographs [7,8] and a recent report summarizes all the published contributions till date (2000) in this domain [9]. Therefore, we shall now present a very brief survey on some of the significant contributions in this area relevant to the problem addressed in this paper and the readers are referred to [9] for an up-to-date survey.

1.1. Related work

In all the research so far in this domain, a criterion that is used to derive optimal solution is as follows. It states that in order to obtain an optimal processing time, it is *necessary and sufficient* that all the processors participating in the computation must stop computing at the same time instant. This condition is referred to as an *optimality principle* in the DLT literature and analytic proof of the assumption for optimal load distribution for bus networks also appears in [10]. The effect of topologies on load distribution was presented in [11] and in this study, the load distribution problem was analyzed on a variety of computer networks such as linear, mesh and hyper-

* Corresponding author.
E-mail: engp0969@nus.edu.sg

cube. Barlas [12] presented an important result concerning an optimal sequencing in a tree network by including the results collection phase. In fact, this is one of the important studies in the DLT literature that demonstrates the performance of the load distribution algorithm when result collection phase is included in the problem formulation. In a recent work reported in [13], a multi-level tree is considered and it is assumed that the load distribution takes place concurrently from a source processor to all its immediate child processors. This is one of the earliest attempts in using a *multi-port* model of communication, a model in which all the incident links on a processor are concurrently used. The use and advantage of a multi-port model was also demonstrated in the study conducted for a two-dimensional mesh network in [14] for scheduling divisible tasks. Theoretical investigation on load partitioning of intensive computations of large matrix-vector products in a multicast bus network was investigated in [15].

To determine the ultimate speedup using DLT analysis, Li [16] conducted an asymptotic analysis for various network topologies. A recent paper [17] introduced simultaneous use of communication links to expedite the communication and the concept of *fractal hypercube* on the basis of *processor isomorphism* to obtain the optimal solution with fewer processors is proposed. With most of the fundamental studies in this area were consolidated in [7,8], research efforts from 1996 particularly started focussing on practical issues such as scheduling divisible loads under real-time deadline constraints [1]. Several practical issues addressed in conventional multiprocessor scheduling problems in the literature were also attempted in the domain of DLT. These studies include, handling multiple loads [18], scheduling divisible loads with arbitrary processor release times in bus networks [19], use of affine delay models for communication and computation for scheduling divisible loads [20,21], scheduling with finite-size buffers [22], and finally, the proposed algorithms were tested using experiments on real-life application problems such as image processing [23], database operations [24], matrix-vector product computations [25]. In [25] rigorous experimental implementation of the matrix-vector products on PC clusters as well as on a network of workstations (NOWs) was carried out. The motivation behind this study is to implement and validate the theoretical findings reported in [15]. In [24], several other applications such as *pattern search*, *file compression*, *joining operation in relational databases*, *graph coloring and genetic search* using the divisible load paradigm, were implemented. Extension of the DLT approach to other areas such as multimedia was first attempted in [26]. In the domain of multimedia, the concept of DLT was cleverly exploited to retrieve a long duration movie from a pool of servers to serve a client site. In section 1.2, we shall highlight our contributions.

1.2. Our contributions

In this paper, we consider the problem of scheduling arbitrarily divisible loads on arbitrary graph networks. Depending on the communication and computation models, one may design a variety of strategies for distributing the load. For instance, in DLT literature all the so far attempts have focussed on using uniport communication model wherein the load fractions from a processor are distributed to others in a one-at-a-time fashion. On the other hand, using a multi-port model, a processor may be able to send the load to all its directly connected neighbors simultaneously [27]. However, provision of supporting hardware for implementing a multi-port model may incur additional cost, as argued in [13]. Further, even if a processor is provided with such a facility, unless it is capable of receiving all the load fractions simultaneously over all the incident links on it, adopting a multi-port model may not be beneficial. Thus, associated with simultaneous load distribution, a processor, in an arbitrary graph, must be provided facilities to simultaneously receive (*gathering*) the data. All these additional demands in resources somewhat make multi-port model practically less adoptable, especially for large volume data handling and processing purposes. In fact, this is one of the implicit reasons why uniport model suits for processing divisible loads.

Although in [13] a multi-port model is attempted, the basic underlying network is indeed a tree network, and hence, every processor will have only one incident link (in-degree) to receive the load. Therefore, although using a multi-port model may deliver an acceptable performance, the authors conclusively had shown that the limitation is essentially due to additional hardware requirements and not with the strategy. On the other hand, since our network is basically an arbitrary network, with every processor having more than one incident link, it is evident that the cost of additional hardware will be prohibitively higher. Thus, in this paper, for processing on arbitrary networks, we shall follow the uniport model adopted in DLT with the assumption that every node has a single processor to process the given load and that it has one front-end that can off-load its communication duties and take care of transferring the loads to processors on a one-at-a-time basis. It may be noted that, on a given graph, adopting a uniport model as explained above, obviously demands determining the best paths to reach the destination processors from a given source. Equivalently, it is mandatory to generate a *spanning tree* that provides best routing paths between processors. Consequently, as a first step, we generate a *minimum cost spanning tree* (MST), denoted as G_{MST} (a generic multi-level unbalanced tree), from the given graph and as a second step, we design optimal scheduling algorithms for the resulting tree networks.

We consider the problem when the load distribution sequence follows a given (fixed) order of distribution in every sub-tree (we refer to a single-level tree network as a sub-tree, hereafter). In the earlier literature, procedures to obtain optimal performance for single-level tree networks have been derived rigorously [7]. We revisit these procedures in a systematic fashion to apply onto a generic multi-level unbalanced tree network. We develop procedures for carrying out the load distribution process by carefully identifying any redundant processor–link pairs (referred to as P–L pairs hereafter), if exists, in the network by applying a criteria referred to as Rule A in the DLT literature [7]. Since the algorithm eliminates any redundant P–L pairs and uses only a subset of effective processors for processing the entire load, we will actually distribute the load on the resulting optimal reduced tree (OPT) network, denoted as G_{opt}, and we refer to this approach as *resource-aware optimal load distribution* (RAOLD) algorithm. As a next step, we extend the study by applying an optimal sequencing theorem proposed in [7,12] and propose a slight variant of our algorithm RAOLD, referred to as RAOLD-OS, in this case. In line with the fixed sequencing case stated above, following the optimal sequencing theorem, we have an optimal tree denoted as $G_{\text{opt}}^{\text{os}}$.

We show the performance of our strategies by rigorous simulation experiments on sparse, medium-dense and dense graph networks generated randomly. We derive certain key features from our simulation studies that can be helpful for network designers and point to certain trade-off relationships on the nature of the underlying network, in terms of its density (number of processors) and their connectivity. Based on rigorous simulation experiments we discuss on the effect of scalability (network size) and when it can become a bottleneck. Although the transformed network is a tree network, for a given graph, with respect to a point of load origination, the depth (number of levels) and the number of processors in a level will differ, and hence, results in performance variation. We rigorously simulate the performance when the point of load origination is different in the network, leading to a useful observation that can be recommended for the designers. We conduct rigorous simulation studies to observe all the above-mentioned features.

The organization of the paper is as follows. In section 2, we present the problem formulation and in the first part of section 3, we first present a method to construct G_{opt} from G_{MST} and propose an optimal distribution algorithm (RAOLD) using Rule A for this optimal reduced tree. Similarly, in the second part of section 3, we propose construction and distribution strategy for $G_{\text{opt}}^{\text{os}}$ using optimal sequencing theorem. Rigorous simulation is attempted in section 4 for different types of graphs with varying link probabilities and different points of load origination. We present discussions for these simulation results in this section as well. Finally, in section 5, we conclude the paper with future possible extensions related to this paper.

2. Problem formulation

In this section, we shall introduce the problem formally and present the required definitions, notations, and terminology that will be used throughout the paper. As mentioned in section 1, we consider the problem of scheduling and processing a divisible load on an arbitrary topology/graph $G = \langle N, E\rangle$ where N denotes the number of processors interconnected via E communication links. The processors and the links are assumed to be heterogeneous in the sense that their respective speeds may not be identical. Thus, the edges have weights corresponding to the speeds of the links. We assume that all the processors in the system have front-ends. This means that each processor can compute and communicate with another processor to which it is connected directly via a link, simultaneously. Further, we consider the load origination on any processor in the network and we are interested to obtain an optimal load distribution such that the total processing time of the entire load is a minimum. Without loss of generality, we shall assume that all the processors in the system are capable of processing the load, that is, the required application to process the load is assumed to be available at all the processors. Thus, in this paper, as mentioned in section 1, we are interested in obtaining an optimal load distribution for the above mentioned case.

We shall now introduce the solution procedure we used to attack the problem. Since all the processors are capable of processing the load, the process of data dissemination must take into account on *how well the load can be distributed* in terms of an apt choice on minimum cost *paths* (cost referring to communication delay, hereafter) used for communicating the data to the processors. In this paper, as a first step, considering the processor at which the load originates as a *root* processor, we attempt to generate a *minimum cost spanning tree* (MST) that spans all the processors from the root processor. Thus, we obtain a MST denoted as G_{MST} from the given G. Note that after this stage, the *costly* links are eliminated, that is those paths that incur the smallest communication delays between any pair of processors are used. It may be noted that at this stage we can adopt any standard known algorithms to generate MST, such as Kruskal's or Prim's algorithm [28]. In our simulation tests, we have used Prim's algorithm to generate G_{MST} by always choosing a path with the smallest weight, which is denoted by the inverse of the link speed. Also, using Fibonacci heaps, Prim's algorithm can be run in time $O(E + N\log N)$. Having obtained G_{MST}, as a second step, one may attempt to use a load distribution strategy described in [29], however, this may not guarantee an optimal solution. Further, in [29], the sequence of load distribution is fixed, from the left end to the right end of the tree starting from the root processor. Also, for this fixed sequence of load distribution, the solution obtained by this procedure will be an optimal solution *if and only if* every subtree in G_{MST} has links that satisfy a condition referred to as **Rule A** in the literature [7]. Rule A proposes a methodology by which redundant P–L pairs (slow processor–link pairs in a given fixed sequence of load distribution) are eliminated in a single-level tree network and generates an *optimal reduced tree* network that yields an optimal solution. In this paper, we attempt to apply this Rule A systematically in a *bottom-up* fashion to G_{MST}, starting from the last level to the root, to obtain an optimal reduced tree, denoted as G_{opt}. Note that *optimal reduced tree* refers to the fact that given a G_{MST}, we apply Rule A to yield a reduced tree that delivers an optimal time performance. We describe these procedures to obtain G_{opt} in section 3. Thus, all the strategies proposed in this paper use the solution methodology so far described.

2.1. Definitions, notations, and some terminology

We now introduce the necessary notations, definitions, and the terminology that will be used throughout the paper:

L	The total amount of load originating at a processor for processing.
p_s	Processor s in the given graph G.
l_{p_s,p_t}	Communication link connecting processors s and t in the graph G.
$p_{x,i}$	This denotes processor i in G_{MST} whose parent is processor x. For the root processor of G_{MST}, we simply denote it as p_0.
$l_{p_{x,i},p_{y,j}}$	Communication link connecting processors $p_{x,i}$ and $p_{y,j}$ in the tree G_{MST}.
$\Sigma(x,i,m+1)$	This is a single-level tree network (sub-tree) defined in G_{MST}, consisting of $(m+1)$ processors, with root processor $p_{x,i}$ and m child processors denoted as $p_{i,1},\ldots,p_{i,k},\ldots,p_{i,m}$. Further, since every child processor has the same parent in this sub-tree, we can conveniently denote the communication link $l_{p_{x,i},p_{i,k}}, k \in (1,\ldots,m)$, connecting $p_{x,i}$ with $p_{i,k}$ simply as $l_{i,k}$. Note that for the single-level tree with root processor p_0, we denote it as $\Sigma(0,m+1)$.
T_{cp}	Time taken to process a unit load by a standard processor.
T_{cm}	Time taken to communicate a unit load on a standard link.
$w_{x,i}$	A constant that is inversely proportional to the speed of processor $p_{x,i}$. Note that, $w_{i,k}$ is the inverse of the *speed* of processor $p_{i,k}$ in $\Sigma(x,i,m+1)$.
$z_{x,i}$	A constant that is inversely proportional to the speed of link $l_{x,i}$. Note that, $z_{i,k}$ is the inverse of the *speed* of link $l_{i,k}$ in $\Sigma(x,i,m+1)$.
$p_{x,\mathrm{eq}(i)}$	An equivalent processor of the entire network $\Sigma(x,i,m+1)$. That is, we replace the entire sub-tree rooted at processor i with its equivalent processor. Note that, processor x now becomes the parent of this equivalent processor. Hence, we denote the respective speed parameter (inverse of the speed) of this equivalent processor as $w_{x,\mathrm{eq}(i)}$. Similarly, the equivalent processor of $\Sigma(0,m+1)$ is denoted as $p_{\mathrm{eq}(0)}$.
$l_{x,\mathrm{eq}(i)}$	An equivalent link which is equivalent to the communication capability of a set of links in $\Sigma(x,i,m+1)$. The respective speed parameter of this equivalent link is denoted by $z_{x,\mathrm{eq}(i)}$.
α	This is defined as a N-tuple and refers to a load distribution, i.e., $\alpha=(\alpha_0,\alpha_{0,1},\ldots,\alpha_{0,m_0},\ldots,\alpha_{x,i},\ldots,\alpha_{g,1},\alpha_{g,2},\ldots,\alpha_{g,m_g})$, where $\alpha_{x,i}$ is the fraction of load assigned to $p_{x,i}$ such that $0 \leqslant \alpha_{x,i} \leqslant 1$ and sum of all the above load fractions amounts to the total load to be processed. Note that $m_0,\ldots,m_g$ reflect the degree of connectivity for each sub-tree.
$\alpha_{x,\mathrm{eq}(i)}$	Load fraction given to the equivalent processor $p_{x,\mathrm{eq}(i)}$ for processing. Note that for $p_{\mathrm{eq}(0)}$ and its equivalent network $\Sigma(0,m+1)$, we denote this value as $\alpha_{\mathrm{eq}(0)}=1$.
$\alpha_{x,i}$	The optimal load fraction assigned to processor $p_{x,i}$ in $\Sigma(x,i,m+1)$. Similarly, for child processor $p_{i,k}$, we denote this value as $\alpha_{i,k}$.
$L_{x,i}$	This is defined as the total amount of load assigned to $p_{x,i}$ in G_{MST}.
$T(\Sigma(x,i,m+1))$	This is defined as the optimal processing time of the assigned load fraction $\alpha_{x,\mathrm{eq}(i)}$ to $\Sigma(x,i,m+1)$. Note that for $\Sigma(0,m+1)$, we denote the optimal processing time as $T(\Sigma(0,m+1))$, which is indeed the processing time of the entire load L.
$T_{x,i}(\alpha)$	The time instant by which processor $p_{x,i}$ stops its computation under the distribution α.
$T(\alpha)$	The total processing time of the entire load under the distribution α. Note that $T(\alpha)=\max\{T_{x,i}(\alpha)\}$ where the maximization is over all the processors in the network.
$T^*(\alpha^*)$	The optimal processing time of a load, which is the minimum processing time to finish the processing of the entire load, using an optimal load distribution α^*.

3. Optimal reduced tree and processing time solution

In this section, using the G_{MST} obtained as described above, we shall derive G_{opt} by systematically applying Rule A. We consider two possible cases of interest. In the first case, in G_{MST} (and in the resulting G_{opt}), without loss of generality, we assume that the load is distributed from the left hand side to the right hand side in every single-level tree network in G_{MST} (and in the resulting G_{opt}). Hence, we assume that the sequence of load distribution is fixed. In the second case, we apply the optimal sequencing theorem proved in [7,12] and study the time performance. For the purpose of continuity, we present the implications of Rule A briefly in the appendix.

Further, in our algorithms, we will be predominantly using the concept of processor–link equivalence. The process of obtaining equivalent processor and link parameters [7] has been described in brief in the appendix. Essentially, in the appendix, we refer to important relationships that will be used in our algorithms for generating equivalent sub-trees. Readers are referred to [7] for further details.

3.1. Construction of G_{opt}

The construction of G_{opt} recursively uses the above described procedure for obtaining an optimal reduced single-level tree network following the three steps of procedure R-O-SLTN described in figure 1. It may be noted that while carrying out step 1 for a single-level tree network rooted at, say processor i, one or more of its child processors could be an equivalent processor of another single-level tree network with i as the parent processor. However, in case, if this equivalent "child" processor is eliminated in step 1, this means that the entire branch beneath processor i will be eliminated from computation process. Figure 8(d) explains this process.

Procedure: **R-O-SLTN(. . .)**

Consider a single-level tree network $\Sigma(x, i, m+1)$ at a level, say j, with m child processors, in the given tree G_{MST}:

Step 1: Apply Rule A as described in the appendix and eliminate any redundant P–L pairs that violated condition (A.1). By doing so, we can get an optimal reduced single-level tree network $\Sigma(x, i, n+1)$ where $n \leqslant m$ [7].

Step 2: Determine the equivalent processor $p_{x,\text{eq}(i)}$, with the equivalent speed parameter $w_{x,\text{eq}(i)}$ given by (A.6), of the single-level tree obtained in step 1.

Step 3: "Replace" the optimal reduced single-level tree in step 1 with the above equivalent processor $p_{x,\text{eq}(i)}$.

Figure 1. Construction of optimal reduced single-level tree network.

Hence, in order to obtain G_{opt}, we use the above procedure R-O-SLTN in a bottom-up fashion, as follows. Starting from the last level, say Q, of G_{MST}, for every single-level tree network we carry out steps 1 through 3 to obtain an equivalent processor. This procedure will be carried out until we reach the root processor/node (top most level) to yield $p_{\text{eq}(0)}$. After applying procedure R-O-SLTN on $\Sigma(0, m+1)$, we can get a single equivalent processor $p_{\text{eq}(0)}$ which is equivalent to the entire optimal reduced tree. Thus, the processing time performance of the entire network G_{opt} is given by $T^*(\alpha^*) = L \cdot w_{\text{eq}(0)} \cdot T_{\text{cp}}$.

On the whole, what we have obtained so far as $p_{\text{eq}(0)}$ is indeed the desired G_{opt}. However, to explicitly obtain the G_{opt} one needs to simply inflate all the equivalent processors at all levels, starting from $p_{\text{eq}(0)}$ onwards. As a part of the computational process, while eliminating the redundant P–L pairs using the procedure R-O-SLTN, it is imperative to track the number of levels that we traverse before reaching the root processor. Hence, if an equivalent processor is eliminated by R-O-SLTN procedure then, since the entire branch beneath this equivalent processor will be eliminated, the number of levels in G_{MST} and the final G_{opt} might be different. Therefore, it is essential that we need to update the number of levels that "survived" in G_{opt} during its construction. Without loss of generality, let us assume that the number of levels of G_{opt} is $Q' \leqslant Q$. This count on the number of levels (of G_{opt}) during the construction of G_{opt} itself is useful for our load distribution process, to be described in section 3.2.

3.2. Load distribution in G_{opt}

The load distribution process in this G_{opt} can be carried out in a systematic fashion as and when we attempt to inflate $p_{\text{eq}(0)}$ from level 1. In general, the idea is to inflate a single-level tree and distribute the load to that tree following the rules of optimal load distribution described in [7]. Thus, as and when we inflate an equivalent processor, load distribution will be carried out. This procedure, referred to as LD, is described in figure 2.

Thus, we observe that during the process of inflating, we carry out the load distribution process simultaneously. Following are some important remarks that should be observed. In step 2 of the above procedure LD, once an equivalent processor at level j is inflated, its child processors constitute the processors (or part of the processors) belonging to the next level ($j+1$). Further, it should be observed that when we inflate an equivalent processor, we mean that we "replace" this equivalent processor with the reduced single-level tree obtained during the construction of original G_{opt} at this processor.

Procedure: **LD(. . .)**

Starting from the equivalent tree $p_{\text{eq}(0)}$ with $\alpha_{\text{eq}(0)} = 1$, we do the following:

Step 1: Inflate $p_{\text{eq}(0)}$ to obtain the reduced single-level tree $\Sigma(0, n+1)$ where $n \leqslant m$, and distribute the load to all the $n+1$ processors in this tree. Note that there are n child processors in $\Sigma(0, n+1)$, corresponding to processors in level 1.

Step 2: For every processor in a level j, $j = 1, 2, \ldots, Q'$, we first verify whether it is an equivalent processor or a leaf processor (a processor with no children in the original G_{MST}). For every equivalent processor i, in this level j, we simply inflate this equivalent processor and optimally distribute the load assigned to this equivalent processor i, by its parent, among the processors that formed this equivalent processor i. Note that this load distribution must be such that all the processors in this single-level tree inflated from equivalent processor i will stop computing at the same time instant, as per the optimality criterion, described by the following equations:

$$\alpha_{i,k} = \alpha_{i,n} \prod_{v=k+1}^{n} f_v, \quad k = 0, 1, \ldots, n-1, \tag{1}$$

where

$$\alpha_{i,n} = \frac{\alpha_{x,\text{eq}(i)}}{1 + \sum_{u=1}^{n} \prod_{v=u}^{n} f_v}, \tag{2}$$

$$f_v = \frac{w_{i,v} + z_{i,v}\delta}{w_{i,(v-1)}}, \tag{3}$$

where $\delta = T_{cm}/T_{cp}$ and $\alpha_{x,\text{eq}(i)}$ is the load assigned to $p_{x,\text{eq}(i)}$ by its parent processor. Also note that when $k = 0$, we refer to the parent processor $p_{x,i}$ with a load fraction $\alpha_{x,i} = \alpha_{i,0}$.

We terminate this step once all the equivalent processors in this level j are inflated.

Step 3: Let $j = j+1$. If level $j \leqslant Q'$, we repeat step 2. Note that $j = Q'$ corresponds to the last level in G_{opt}.

Figure 2. Load distribution procedure in G_{opt}.

From the above description we note that our proposed algorithm recommends a two phase approach. In the first phase, the algorithm attempts to pool the effective number of processors (resources) that can be used for processing. In order to be aware of the resources, we first attempt to invoke procedure R-O-SLTN as shown in figure 1, starting from the last level of G_{MST}. This process percolates upwards till we reach the root processor, at which we obtain $p_{\text{eq}(0)}$, which is the equivalent processor of the entire tree. Thus, we first "shrink" the original G_{MST} to reduce to an equivalent processor. In the

```
Procedure: **RAOLD**(...)
  **Phase I:** Construction of an optimal reduced equivalent tree:
    *Note*: Phase I involves the following procedures: (a) eliminating redundant P–L pairs; (b) generating equivalent processors; (c) building $G_{\text{opt}}$; and (d) counting the number of levels $Q'$ in $G_{\text{opt}}$.
      for ($j = Q$; $j > 0$; $j - -$) /* $Q$: total number of levels in $G_{\text{MST}}$
      {
        for ($r = 1$; $r \leqslant R$; $r + +$) /* $R$: total number of sub-trees at level $j$
          { $p_{x,\text{eq}(i)}$ = R-O-SLTN(...); /* Refer to figure 1 }
      }

  **Phase II:** Load distribution for a fixed sequence:
    *Note*: Phase II involves scheduling the load $L$ among all the processors $p_{x,i} \in G_{\text{opt}}$.
      $\alpha^* = \text{LD}(L)$, from which we have $\alpha^*_{x,i}$ for each participating processor;
      /* Refer to figure 2

**Optimal Load Fractions:** $L^*_{x,i} = \alpha^*_{x,i} \cdot L$;
**Optimal Processing Time:** $T^*(\alpha^*) = T(\Sigma(0, m+1)) = T^*_{x,i}(\alpha^*), \forall p_{x,i} \in G_{\text{opt}}$. /* As given in the appendix and as per the optimality principle.
```

Figure 3. Algorithm RAOLD.

second phase, we invoke procedure LD shown in figure 2 to disseminate the entire load L, starting from the root processor. Since the algorithm thrives to determine the equivalent number of processors (resources) that can be used for processing the entire load, we refer to this approach as *resource-aware optimal load distribution* (RAOLD) algorithm in this paper. Figure 3 presents the corresponding pseudo-code for the complete workings of this two-phase approach.

3.3. Using G_{MST} with optimal load sequencing

In this section, we attempt to apply the optimal sequencing theorem to distribute the load in the G_{MST}. For the purpose of continuity, we state the optimal sequencing theorem quoted in [7] for a single-level tree network.

Theorem 1 (Optimal sequence). In a single-level tree network $\Sigma(x, i, m+1)$, in order to achieve minimum processing time, the sequence of load distribution by the root processor $p_{x,i}$ should follow the order in which the link speeds $(\frac{1}{z_{i,k}}, k = 1, 2, \ldots, m)$ decrease.

The significance of the theorem is that, it essentially provides a necessary and sufficient condition for a sequence of load distribution to be optimal. According to this condition, the load should be distributed first through the fastest link, then through the next fastest link, and so on until the slowest link is assigned the last load fraction. Thus, by adopting such a sequence of load distribution according to the speeds of the links, an optimal processing time will be achieved.

The above theorem, in our context, is applied as follows. The workings of the algorithm for this case is essentially similar to RAOLD algorithm, however, the key difference lies in the first phase. In the first phase, since optimal sequencing does not eliminate any of the P–L pairs, Rule A need not be applied, thus reducing the computational complexity. Hence, after obtaining G_{MST}, we systematically apply theorem 1 to identify an optimal sequence in every single-level tree network, starting from the last level to the root processor. This is important to identify as every parent processor will follow this optimal sequence while distributing the load. For example, in a single-level tree at a level j, a processor i with 4 child processors $(p_{i,1}, l_{i,1})$, $(p_{i,2}, l_{i,2})$, $(p_{i,3}, l_{i,3})$, $(p_{i,4}, l_{i,4})$ correspond to the processor–link pairs starting from the left hand side to the right hand side. Suppose $z_{i,1} = 1.0$, $z_{i,2} = 0.5$, $z_{i,3} = 0.8$ and $z_{i,4} = 1.2$, then the parent processor of this single-level tree network would identify the optimal sequence as $(p_{i,2}, l_{i,2})$, $(p_{i,3}, l_{i,3})$, $(p_{i,1}, l_{i,1})$, $(p_{i,4}, l_{i,4})$, respectively. Thus, every parent processor will apply theorem 1 and obtain the optimal sequence. Another important point to be noted here is that under optimal sequencing, Rule A discussed in the appendix will be automatically satisfied by every link in a single-level tree network, and hence, none of the P–L pairs will be eliminated [7]. Hence, it is expected that the final optimal processing time obtained by optimal sequencing will be better than that obtained for any arbitrary sequence described in the previous section.

The G_{opt} in this case, is identical to that of G_{MST} except that the load distribution will follow an optimal sequence. We denote G_{opt} following an optimal sequence as $G^{\text{os}}_{\text{opt}}$. Thus, as in the previous section, we again obtain $p^{\text{os}}_{\text{eq}(0)}$, by systematically "collapsing" all the single-level tree networks from the last level to the root processor, following the procedure OP-SEQ in figure 4.

Thus, using the above procedure, we can obtain $p^{\text{os}}_{\text{eq}(0)}$, by systematically "collapsing" all the single-level tree networks from the last level to the root processor. As a result, what we have obtained so far as $p^{\text{os}}_{\text{eq}(0)}$ is indeed the desired tree $G^{\text{os}}_{\text{opt}}$. However, to explicitly obtain the $G^{\text{os}}_{\text{opt}}$ one needs to simply inflate all the equivalent processors at all the levels, starting from $p^{\text{os}}_{\text{eq}(0)}$ onwards. Note that step 1 in R-O-SLTN is avoided in procedure OP-SEQ since we know that Rule A will not eliminate any P–L pairs in a single-level tree network

Procedure: **OP-SEQ(...)**

Consider a single-level tree network $\Sigma(x, i, m+1)$ at a level, say j, with m child processors, in the given tree G_{MST}:

Step 1: Identify the optimal sequence of load distribution by applying theorem 1. Determine the equivalent processor $p^{\text{os}}_{x,\text{eq}(i)}$, with the equivalent speed parameter $w^{\text{os}}_{x,\text{eq}(i)}$ given by (A.6), of this single-level tree.

Step 2: "Replace" the single-level tree in step 1 with the above equivalent processor $p^{\text{os}}_{x,\text{eq}(i)}$.

Figure 4. Determination of equivalent processor of single-level tree networks using theorem 1.

following an optimal sequence of load distribution. Further, note that while "collapsing" every single-level tree network $\Sigma(x, i, m+1)$, we find the equivalent speed parameter of that single-level tree using the optimal sequence, which we denote as $w^{\text{os}}_{x,\text{eq}(i)}$. We represent the equivalent processor of the entire tree as $p^{\text{os}}_{\text{eq}(0)}$, to stress that, in this case, we use theorem 1, to avoid any confusion. Thus, after obtaining $p^{\text{os}}_{\text{eq}(0)}$, we use the following procedure LD-OS described in figure 5 to distribute the load. Figure 6 presents the corresponding pseudo-code for the complete workings of the entire algorithm proposed here, referred to as RAOLD-OS.

The following numerical example illustrates the workings of algorithms RAOLD and RAOLD-OS in obtaining an optimal load distribution.

Example 1. We consider an arbitrary graph network G with 9 processors interconnected via 17 communication links, as shown in figure 9(a). We assume the following parameters. The processor speed parameters are $w_1 = 2.0$, $w_2 = 3.0$, $w_3 = 1.5$, $w_4 = 1.2$, $w_5 = 1.5$, $w_6 = 1.0$, $w_7 = 2.0$, $w_8 = 1.0$, $w_9 = 1.0$ and the respective link speed parameters are marked in the graph. Further, we let $L = 100$. Note that the load originates at processor p_9 in the given graph G. Hereafter, we use the notations defined for G_{MST} for processors and links. For the ease of understanding and to avoid any confusion, in figure 9(b), we identify the corresponding processors and links in G in brackets.

In this example, we compute load distributions for G_{MST}, G_{opt}, and $G^{\text{os}}_{\text{opt}}$ by employing the proposed algorithms, as shown in figures 9 (b)–(d). As shown in table 1, we have the respective processing times $T(G_{\text{MST}})$, $T^*(G_{\text{opt}})$, and $T^*(G^{\text{os}}_{\text{opt}})$ for these three cases, respectively. Note that $T(G_{\text{MST}}) = 58.12$ (refer to its timing diagram in figure 10) is computed for comparison purposes. This is done by directly scheduling the entire load on the original G_{MST} without applying optimal sequencing theorem and without eliminating any redundant P–L pairs.

Using algorithm RAOLD, we eliminate redundant P–L pairs $(p_{0,1}, l_{0,1})$, $(p_{2,1}, l_{2,1})$, and $(p_{2,2}, l_{2,2})$ from G_{MST} and distribute the load on the resulting G_{opt}. Thus, every processor was given its optimal load fraction: for the redundant processors $L^*_{0,1} = 0.00$, $L^*_{2,1} = 0.00$, $L^*_{2,2} = 0.00$, and

Procedure: **LD-OS(...)**

Starting from the equivalent tree $p^{\text{os}}_{\text{eq}(0)}$ with $\alpha^{\text{os}}_{\text{eq}(0)} = 1$, we do the following:

Step 1: Inflate $p^{\text{os}}_{\text{eq}(0)}$ to obtain a single-level tree $\Sigma(0, m+1)$ and distribute the load to all the $m+1$ processors in this tree following the optimal sequence. Note that there are m child processors in $\Sigma(0, m+1)$, corresponding to processors in level 1.

Step 2: For every processor in a level j, $j = 1, 2, \ldots, Q$, we first verify whether it is an equivalent processor or a leaf processor (a processor with no children in the original G_{MST}). For every equivalent processor i, in this level j, we simply inflate this equivalent processor and optimally distribute the load assigned to this equivalent processor i, by its parent, following an optimal sequence, among the processors that formed this equivalent processor i. Note that this load distribution must be such that all the processors, say $(m+1)$, in this single-level tree inflated from equivalent processor i will stop computing at the same time instant, as per the optimality criterion, described by the following equations.

$$\alpha^{\text{os}}_{i,k} = \alpha^{\text{os}}_{i,m} \prod_{v=k+1}^{m} f_v, \quad k = 0, 1, \ldots, m-1, \tag{4}$$

where

$$\alpha^{\text{os}}_{i,m} = \frac{\alpha^{\text{os}}_{x,\text{eq}(i)}}{1 + \sum_{u=1}^{m} \prod_{v=u}^{m} f_v}, \tag{5}$$

where f_v is as defined in (3) and $\alpha^{\text{os}}_{x,\text{eq}(i)}$ is the load assigned to $p^{\text{os}}_{x,\text{eq}(i)}$ by its parent processor. Also note that when $k = 0$, we refer to the parent processor $p_{x,i}$ with a load fraction $\alpha^{\text{os}}_{x,i} = \alpha^{\text{os}}_{i,0}$.

We terminate this step once all the equivalent processors in this level j are inflated.

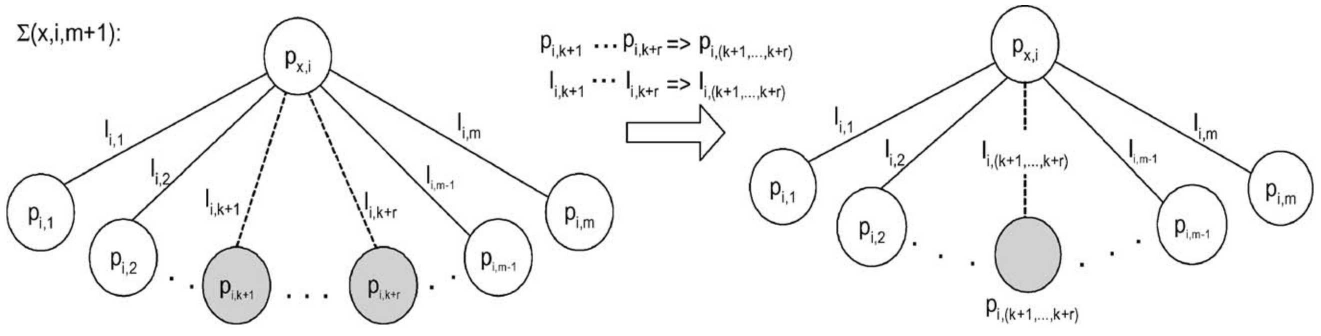
Step 3: Let $j = j + 1$. If level $j \leqslant Q$, we repeat step 2. Note that $j = Q$ corresponds to the last level in $G^{\text{os}}_{\text{opt}}$.

Figure 5. Load distribution procedure in $G^{\text{os}}_{\text{opt}}$.

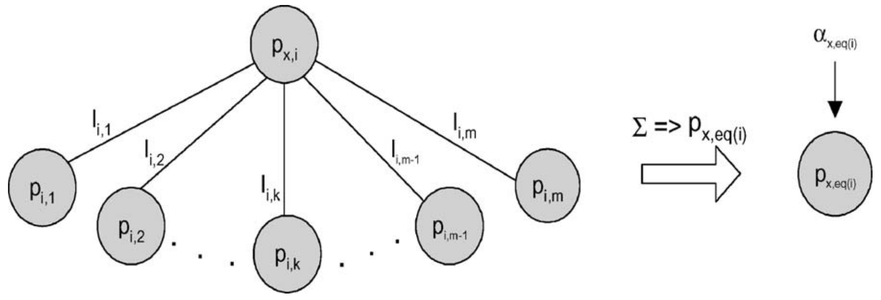
for all the participating processors, we have $L^*_0 = 51.50$, $L^*_{0,2} = 15.85$, $L^*_{0,3} = 13.39$, $L^*_{2,3} = 11.89$, $L^*_{3,1} = 3.35$, and $L^*_{3,2} = 4.02$. Thus, starting from the root processor p_0, the scheduler at every parent processor shall distribute these load fractions to their respective child processors, to obtain the minimum processing time. Note that the entire distribution procedure should be carried out in a way that is consistent with the network structure. This means that, for instance, at first, p_0 will distribute $(L^*_{0,2} + L^*_{2,3})$ as a whole to $p_{0,2}$ first, $(L^*_{0,3} + L^*_{3,1} + L^*_{3,2})$ as a whole to processor $p_{0,3}$ next, and compute its own portion L^*_0 concurrently. Next, the respective schedulers at parent processors $p_{0,2}$ and $p_{0,3}$ will do identical jobs as p_0. For instance, after the above load $(L^*_{0,3} + L^*_{3,1} + L^*_{3,2})$ arrives at $p_{0,3}$, it then transfers $L^*_{3,1}$, and $L^*_{3,2}$ to its child processors $p_{3,1}$ and $p_{3,2}$, respectively. Thus, the optimal processing time for this load distribution is then given by, $T^*(G_{\text{opt}}) = 51.51$. Timing diagram shown in figure 11 demonstrates the exact load distribution process and shows *how* and *when* these load fractions are to be com-

```
Procedure: RAOLD-OS(...)
  Phase I: Construction of optimal equivalent tree:
    Note: Phase I involves the following procedures: (a) identifying an optimal sequence; (b) generating equivalent processors; and
    (c) building $G_{\text{opt}}^{\text{os}}$.
      for ($j = Q$; $j > 0$; $j--$) /* $Q$: total number of levels in $G_{\text{MST}}$
        {
          for ($r = 1$; $r \leqslant R$; $r++$) /* $R$: total number of sub-trees at level $j$
            { $p_{x,\text{eq}(i)}$ = OP-SEQ(...); /* Refer to figure 4 }
        }

  Phase II: Load distribution with optimal sequence:
    Note: Phase II involves scheduling the load $L$ among all the processors $p_{x,i} \in G_{\text{opt}}^{\text{os}}$.
      $\alpha^* = \text{LD-OS}(L)$, from which we have $\alpha^*_{x,i}$ for each processor;
      /* Refer to figure 5

Optimal Load Fractions: $L^*_{x,i} = \alpha^*_{x,i} \cdot L$;
Optimal Processing Time: $T^*(\alpha^*) = T(\Sigma(0, m+1)) = T^*_{x,i}(\alpha^*), \forall p_{x,i} \in G_{\text{opt}}^{\text{os}}$. /* As given in the appendix and as per the optimality
principle.
```

Figure 6. Algorithm RAOLD-OS.



(a)



(b)

Figure 7. Two cases of processor equivalence: (a) equivalence of a set of P–L pairs used in Rule A; (b) equivalence of an entire single-level tree.

Table 1
Load distribution results of example 1.

	L_0	$L_{0,1}$	$L_{0,2}$	$L_{0,3}$	$L_{2,1}$	$L_{2,2}$	$L_{2,3}$	$L_{3,1}$	$L_{3,2}$	
G_{MST}	58.11	14.53	8.94	7.55	3.83	1.80	1.08	1.89	2.27	$T(G_{\text{MST}}) = 58.12$
G_{opt}	51.50	0.00	15.85	13.39	0.00	0.00	11.89	3.35	4.02	$T^*(G_{\text{opt}}) = 51.51$
$G_{\text{opt}}^{\text{os}}$	44.07	2.58	6.87	24.48	1.47	0.69	5.15	4.90	9.79	$T^*(G_{\text{opt}}^{\text{os}}) = 44.07$

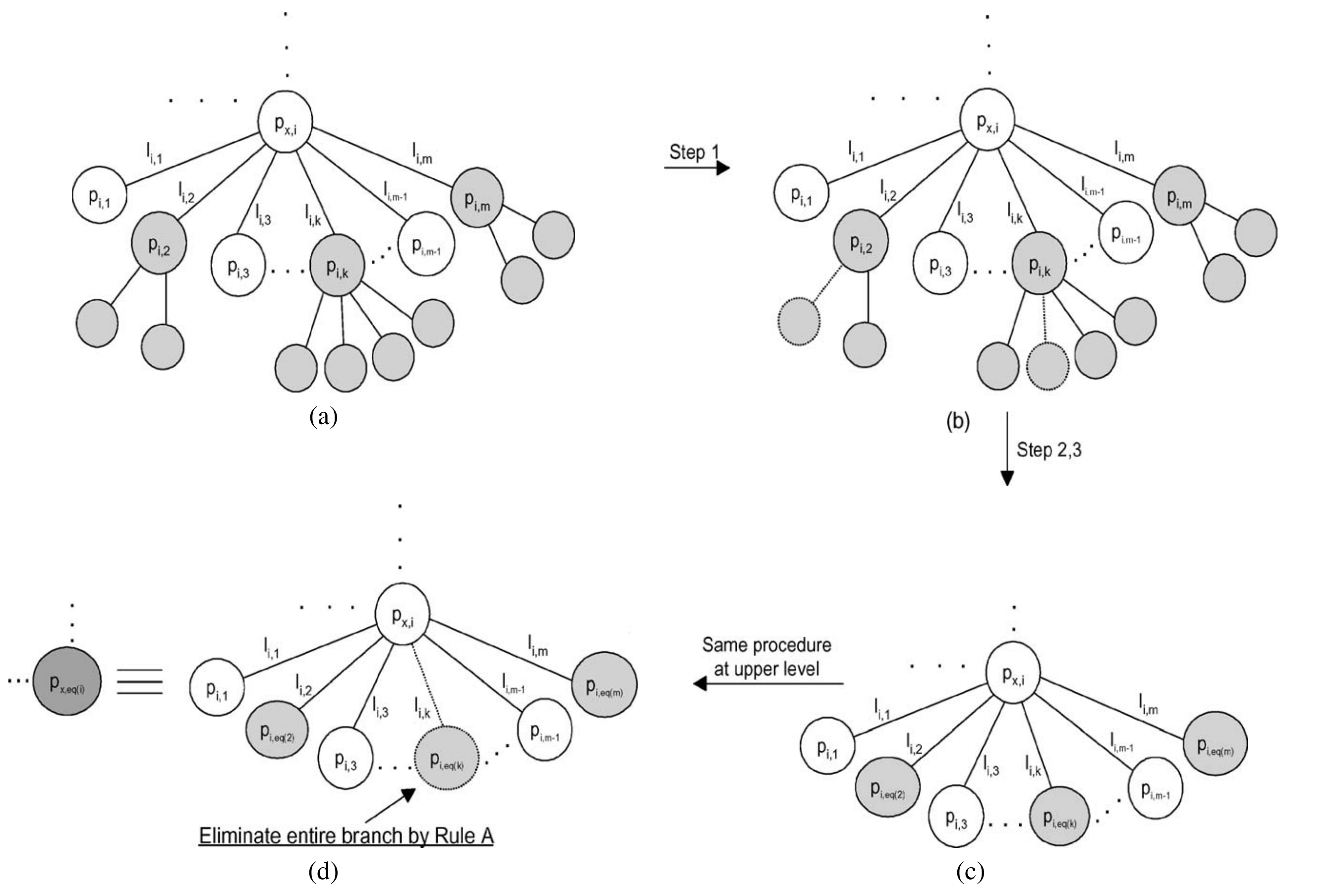

Step 1
Step 2,3
Same procedure at upper level
Eliminate entire branch by Rule A
(a)
(b)
(c)
(d)

Figure 8. Working style of procedure R-O-SLTN.

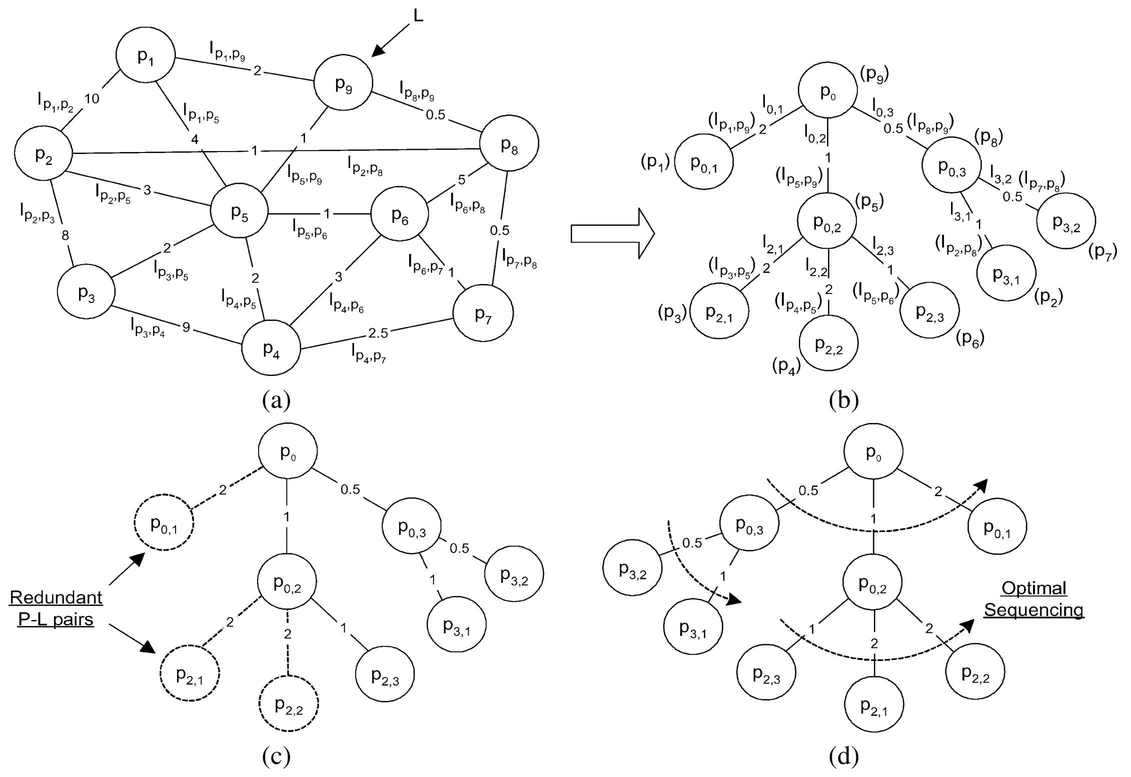


Figure 9. Example 1: (a) An arbitrary graph network G; (b) minimum cost spanning tree G_{MST}; (c) optimal reduced tree G_{opt}; (d) $G^{\text{os}}_{\text{opt}}$ with optimal sequencing.

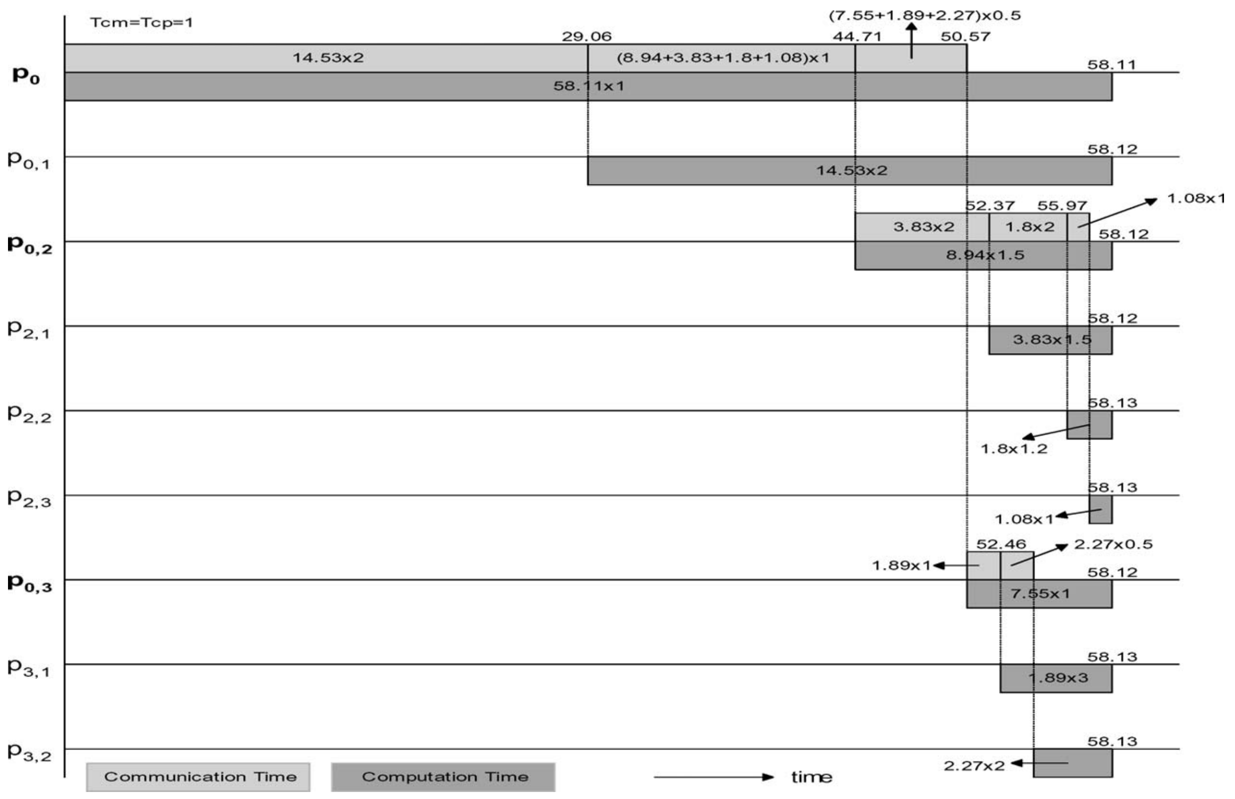


Figure 10. Timing diagram for load distribution in G_{MST} (example 1).

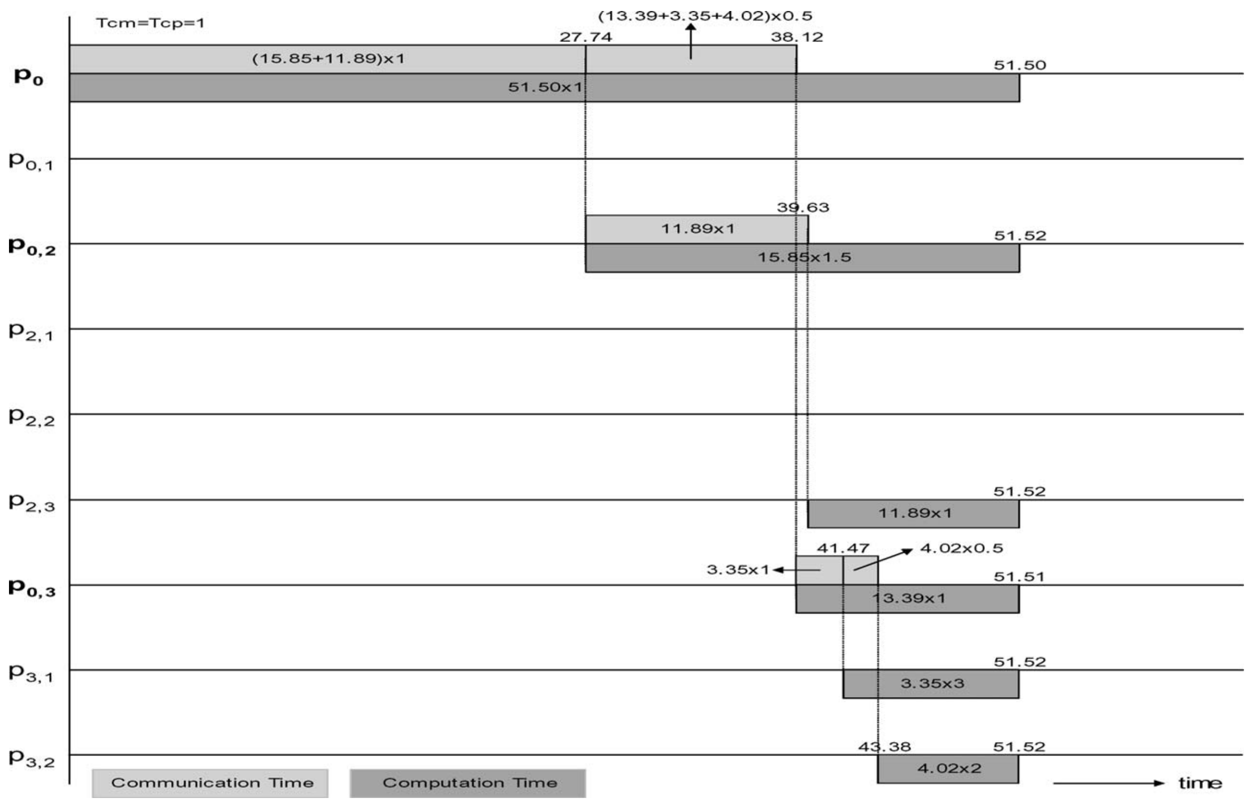


Figure 11. Timing diagram for load distribution in G_{opt} using RAOLD (example 1).

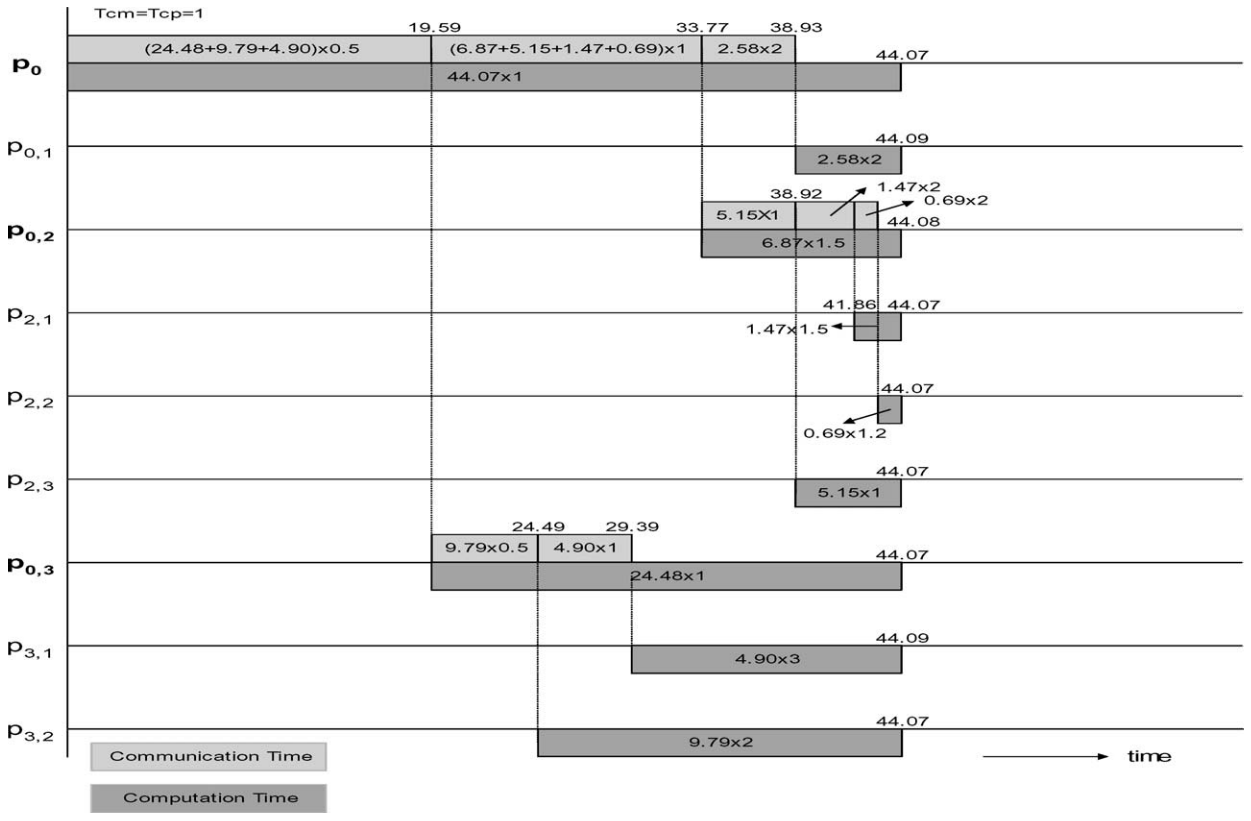


Figure 12. Timing diagram for load distribution in $G_{\text{opt}}^{\text{os}}$ using RAOLD-OS (example 1).

municated within the network and computed at each processor.

Similarly, using algorithm RAOLD-OS, we construct $G_{\text{opt}}^{\text{os}}$ shown in figure 9(d) by applying the theorem of optimal sequencing and distribute the load on this tree. Thus, in this case, every processor was utilized and has been assigned its optimal load fraction: $L_0^* = 44.07$, $L_{0,1}^* = 2.58$, $L_{0,2}^* = 6.87$, $L_{0,3}^* = 24.48$, $L_{2,1}^* = 1.47$, $L_{2,2}^* = 0.69$, $L_{2,3}^* = 5.15$, $L_{3,1}^* = 4.90$, and $L_{3,2}^* = 9.79$. As can be observed from the timing diagram (figure 12), we obtain the minimum processing time as $T^*(G_{\text{opt}}^{\text{os}}) = 44.07$.

4. Simulation results and discussion

In this section, we shall study the performance of all the strategies proposed above for several situations by means of extensive simulations. We will highlight and discuss all the important issues observed from our simulation results. For a complete understanding of the algorithms, we compare the measurements on the processing time $T(G_{\text{MST}})$, $T(G_{\text{opt}})$, and $T(G_{\text{opt}}^{\text{os}})$ for each case. Note that $T(G_{\text{opt}})$ and $T(G_{\text{opt}}^{\text{os}})$ denote the total processing time for algorithms RAOLD and RAOLD-OS presented in section 3, respectively. Also, as mentioned earlier, $T(G_{\text{MST}})$ is calculated for comparison purposes by direct application of the load scheduling on the original G_{MST} without applying optimal sequencing theorem and without eliminating any redundant P–L pairs.

4.1. Arbitrary graph generation and simulation set-up

We now describe how graphs and other speed parameters required for our algorithms are generated. To reflect real-life situations, the graph generation procedure is made completely non-deterministic. Without loss of generality, we set processor p_0 in the network as the root processor. Each graph network used in our experiments is arbitrarily generated with a given number of processors. Further, we use a parameter P_{link} to denote the degree of connectivity, which can also be referred to as *link probability*. The processor speed and the link speed parameters are chosen using a uniform probability distribution in the range [0.01, 10]. We let $L = 10^8$ in all the experiments.

Further, we consider several types of networks such as *sparse*, *medium-dense*, and *dense* so as to capture the performance of our algorithms. It may be noted that the parameter P_{link} defined above allows us to generate these respective networks.

4.2. Effect of network scalability

As a first attempt, we now study the effect of network scalability on the performance of the algorithms. By varying the link probability (P_{link}) as 30, 60 and 90%, we have sparse, medium-dense, and dense networks, respectively. That is, the connectivity is varied from 30 to 90% to capture different types of graphs mentioned above. Also, in each type of the graph, we vary the number of processors from 10 (considered to be a small-size graph) to 300 (considered to be a large-size graph). Note that the sparsity of a graph is reflected by altering the link probability and not with respect to the number of processors in the network. Thus, we can have a large number of processors connected in a sparse manner and on the other hand, we can have a small number of processors with very high connectivity. In fact, this will be an essential step to reflect the real-life situations. Thus, the above process of generating the graphs allows us to study the effect of scalability of the network size with respect to the size of the load to be processed by the system.

As can be observed from tables 2, 3, and 4, for different P_{link} values, the processing time tend to decrease as the number of processors increases. Now, for a fixed P_{link}, it should be observed that the number of communication links increases as the number of processors are increased. Thus, for a given amount of load, it is naturally more efficient to distribute the load on a larger size graph with more P–L pairs available to utilize. Furthermore, it can be noted that, in each of the above tables, $T(G_{\text{opt}})$ always outperforms $T(G_{\text{MST}})$ and $T(G_{\text{opt}}^{\text{os}})$ always outperforms $T(G_{\text{opt}})$ under any network settings, as expected. However, this advantage does not carry far, since when there are no redundant P–L pairs in G_{MST}, we have $T(G_{\text{opt}}) = T(G_{\text{MST}})$, as can be seen with our results in table 3.

Another interesting phenomena that can be observed is on the influence of the number of redundant P–L pairs eliminated on the processing time. Referring to the fourth column of table 2, the number of eliminated P–L pairs can be as small as 2 with 50 processors in the original network, for instance, while the resulting time performance is significantly improved by 10.74% for this case. Also, we observe that when the number of processors is 100, although our algorithm utilizes only 5 processors on the whole for processing the entire load (95 P–L pairs are eliminated), we are still able to achieve a considerable gain by 1.88%. In fact, comparing the exact magnitudes of the processing times, we observe that there is a significant improvement in performance. Indeed, regardless of how many redundant P–L pairs are to be eliminated, RAOLD always attempts to render the most efficient G_{opt} for the scheduling. Thus, as far as network scalability is concerned, our study clearly elicits the following useful observations. In order to minimize the processing time, it may not be beneficial to choose a larger network (in terms of number of processors) inadvertently, since, eventually more P–L pairs may get eliminated from processing by our algorithms.

Table 2
Simulation results for load distribution in sparse graphs. $P_{\text{link}} = 30\%$.

Number of processors	Number of links	$T(G_{\text{MST}})$	Number of P–L pairs eliminated in G_{MST}	$T(G_{\text{opt}})$	$T(G_{\text{opt}}^{\text{os}})$
10	10	470473718.6432	2	470151090.6219	467131328.5828
50	357	188237082.9582	2	168016874.7902	163955390.4533
100	1547	74920123.8155	95	73512899.8756	70972865.8199
150	3477	69116181.1352	24	69101923.7041	67866021.3947
200	6135	66770297.2889	16	64354526.9966	59279567.0033
300	13827	28086963.2959	20	28086951.3750	27911278.6055

Table 3
Simulation results for load distribution in medium-dense graphs. $P_{\text{link}} = 60\%$.

Number of processors	Number of links	$T(G_{\text{MST}})$	Number of P–L pairs eliminated in G_{MST}	$T(G_{\text{opt}})$	$T(G_{\text{opt}}^{\text{os}})$
10	22	50760453.9394	2	50760442.0185	50743854.0459
50	755	40193861.7229	7	37953642.0107	27336654.0670
100	3057	29176443.8152	0	29176443.8152	28617811.2030
150	6839	55762463.8081	14	55476880.0735	53347784.2808
200	12171	12818944.4542	2	12818856.5373	11675956.1002
300	27461	10500698.5366	3	10495521.8732	9452676.7731

Table 4
Simulation results for load distribution in dense graphs. $P_{\text{link}} = 90\%$.

Number of processors	Number of links	$T(G_{\text{MST}})$	Number of P–L pairs eliminated in G_{MST}	$T(G_{\text{opt}})$	$T(G_{\text{opt}}^{\text{os}})$
10	38	175574374.1989	4	167196905.6129	149416160.5835
50	1107	57454627.7523	1	57453489.3036	57107806.2058
100	4508	45715668.7975	4	45597612.8578	43640163.5408
150	10149	12205053.1209	19	12070950.8657	10557768.4939
200	18102	17606242.0011	2	17604456.8419	16782523.6917
300	40780	3249463.4390	3	3249386.6980	3202208.5041

Table 5
Simulation results for load distribution in graphs with different P_{link}. Number of processors = 50.

P_{link}	Number of links	$T(G_{\text{MST}})$	Number of P–L pairs eliminated in G_{MST}	$T(G_{\text{opt}})$	$T(G_{\text{opt}}^{\text{os}})$
10%	127	175372803.2112	30	173523497.5815	136172842.9794
30%	357	188237082.9582	2	168016874.7902	163955390.4533
50%	637	57956451.1776	16	57571679.3537	54263865.9477
70%	881	73335516.4528	6	73249876.4992	67480820.4174
90%	1107	57454627.7523	1	57453489.3036	57107806.2058
100%	1225	35981160.4023	0	35981160.4023	35587531.3282

On the other hand, even with a small size network, with few P–L pairs getting eliminated, we may still obtain a better performance than the former case. Thus, merely scaling up the network size may not fetch a desirable performance.

Further, we compare the performance among these sparse and dense networks from a broader perspective. From our experiments, we observe that the time performance of our algorithms differ with respect to the link probabilities, and hence, we may naturally attribute this difference to the density of the network. Consequently, it is of natural interest to answer the following question: *Does a higher link probability always guarantee a better time performance?* To testify this observation, in the next part of our simulation tests, we shall study the relationship between the time performance and the link probabilities, in detail.

4.3. Effect of network connectivity

As a second attempt, we vary the link probability of the network to observe the performance under a fixed number of processors (50 in our experiment). Note that in order to guarantee the generated graph is a connected graph, the value of P_{link} cannot be close to zero. When $P_{\text{link}} = 100\%$, we have a complete graph network in which every processor is connected with any other processors by a direct link. Thus, we study the effect of network connectivity by varying the P_{link} in the range $[0.1, 1]$, as shown in table 5. Although one may expect that as P_{link} increases, the processing time tend to decrease, our experiments show that this may not be true in general. The former obvious expectation is due to the fact that since the link probability, on an average, determines the number of links, with a higher P_{link} value, more links can be considered for choosing the best routes during the construction of G_{MST}. Thus, a higher link probability can ensure us to construct a more efficient G_{MST} to schedule the load. However, a larger value of P_{link} does not influence the performance of our algorithms, as ultimately the processing time depends on how many effective P–L pairs participate in the processing. Thus, by increasing the degree of connectivity all that we can expect is to minimize the processing time, since probability of more P–L pairs participating in the processing also increases. However, performance improvement may not be guaranteed (see table 5 when $0.3 < P_{\text{link}} < 0.7$).

4.4. Effect of load origination

As mentioned in section 1, the incoming divisible load is assumed to originate at any processor in the network. Thus, in a realistic situation, it is possible that any processor in G can become a root processor from which G_{MST} will be constructed. Therefore, it is apparent that the place at which load originates will influence the time performance since choosing different root processors will engender different G_{MST}s. To study this effect of load origination, by varying the point of load origination (root processor), we observe the time performance of a graph network with a fixed number of processors (= 10) and link probability (= 50%), as shown in figure 13(a). As can be observed from table 6, the processing time is shown to be quite nonuniform when the point of origination is varied among processors p_1 to p_{10} in G. However, when the point of load origination is p_4, we observe a minimum processing time for $T(G_{\text{MST}})$, $T(G_{\text{opt}})$ and $T(G_{\text{opt}}^{\text{os}})$, respectively, when compared to all the other cases. For instance, the time performance obtained for the above respective tree structures (G_{MST}, G_{opt}, and $G_{\text{opt}}^{\text{os}}$) are improved almost by a factor of 150 times compared with the respective cases when the load origination is at p_{10}. Now, to understand on how this gain results, we compare the G_{MST}s that are generated when p_4 and p_{10} are the points of load origination (see figures 13(b) and (c)). Comparing these cases, we observe that the total processing time is optimal when p_4 is the point of origination. Thus, we observe that for this case, the root processor p_4 is the fastest processor that distributes the load to the entire tree, and hence, we may expect that it may be beneficial if the processing load is scheduled by the fastest processor in the network. Although one has no choice in deciding the point of load origination in real-life situations, the implications of this result are as follows. Suppose that if there are k processors demanding computation of divisible loads. Then the above example clearly elicits the fact that by initiating the processing of a load that is from the fastest processor first is beneficial in optimizing the time performance of all the loads, taken one at a time, from the fastest to the slowest processor location. Although this is an observation, a formal proof is yet to be attempted. Thus, we observe that the time performance has an important relationship with the point of load origination.

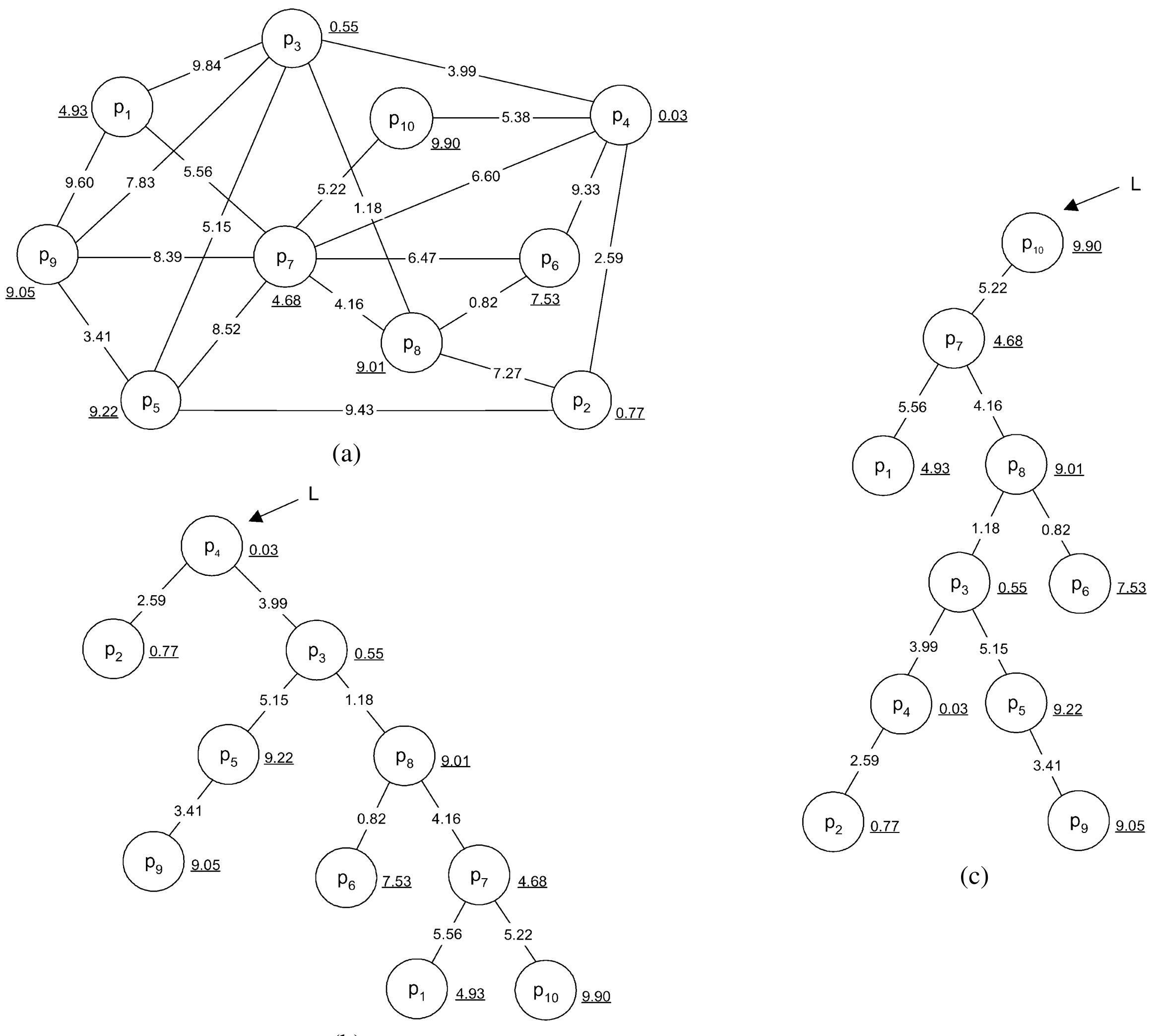


Figure 13. Effect of load origination: (a) An arbitrary graph network G with 10 processors; (b) G_{MST} rooted at p_4; (c) G_{MST} rooted at p_{10}.

Table 6
Simulation results for load distribution with different points of load origination. Number of processors = 10, $P_{\text{link}} = 50\%$, number of links = 20.

Root processor	$T(G_{\text{MST}})$	Number of P–L pairs eliminated in G_{MST}	$T(G_{\text{opt}})$	$T(G_{\text{opt}}^{\text{os}})$
1	304872012.1384	0	304872012.1384	304778933.5251
2	59509295.2251	2	59509289.2647	59509289.2647
3	48341730.237007	4	48315235.9724	44537860.1551
4	2968935.2959	2	2968928.5904	2968913.1305
5	341460156.4407	2	341455507.2784	313949751.8539
6	167124700.5463	0	167124700.5463	167113327.9800
7	248732066.1545	1	242934322.3572	233902883.5297
8	127580666.5421	0	127580666.5421	123482811.4510
9	391724395.7520	2	391721725.4639	391253137.5885
10	434861087.7991	1	434647464.7522	430037784.5764

4.5. Derivation of complexities of the algorithms

Considering an arbitrary graph $G = \langle N, E\rangle$, we shall now quantify the complexities of the algorithms proposed in this paper. As a first step of our algorithms, we have used Prim's algorithm to generate G_{MST} in our simulation tests. Thus, it takes us $O(E + N\log N)$ steps to transform the problem from G to G_{MST}. Further, at the scheduling stage, in our calculations, we shall compute the number of steps involved during the construction of $G_{\text{opt}}/G_{\text{opt}}^{\text{os}}$ (Phase I) and the load distribution procedure (Phase II), respectively. Assuming that there are m processors in every sub-tree and that there are R sub-trees in every level, with a total number of Q levels in the entire tree network G_{MST}, the complexity to construct G_{opt} can be readily seen as $O((m+1)RQ) = O(N+RQ)$ steps, where N is the total number of processors in the network. The factor m comes because of the fact that it consumes $O(m)$ steps per sub-tree for validating Rule A (and the computation of equivalent processor–link is done in $O(1)$ using (A.6) for each sub-tree). For the load distribution phase, the complexity would be $O(N)$ steps. Thus, the total complexity of RAOLD is $O(E+N\log N+2N+RQ)$. Considering $R \leqslant N$, $Q \leqslant N$, and $2N \leqslant N \log N$ (for $N \geqslant 4$), we can simplify the complexity to $O(E+N^2)$.

Similarly, for algorithm RAOLD-OS using optimal load sequencing, we follow the same argument as above to calculate the complexity. It may be noted that the key difference of the two algorithms lies in the procedure to construct the optimal tree. Taking into account of the sequencing, the complexity to construct $G_{\text{opt}}^{\text{os}}$ is given by $O((1+\log m)\cdot RQ) = O(RQ+RQ\log m)$ (consuming $O(\log m)$ steps for sequencing per sub-tree with m processors). Thus, the total complexity of RAOLD-OS is given by $O(E+N\log N+N+RQ\log m+RQ)$. Similar to the above case, since $m \leqslant N$ and $RQ \leqslant RQ\log m$, we can simplify the complexity as $O(N^2\log N)$.

5. Conclusions

In this paper, we investigated the problem of scheduling a divisible load on a given arbitrary graph network using a uniport communication model. We transform the problem into a multi-level unbalanced tree network that consists of minimum communication delay paths to transfer the load fractions among the processors. In the first part of our work, we have proposed a two phase algorithm RAOLD which identifies and eliminates any existing redundant P–L pairs in the network and constructs an optimal tree G_{opt} to obtain the optimal processing time, for a fixed sequence of load distribution. Further, following the same approach, we proposed a variant to our RAOLD algorithm, referred to as RAOLD-OS, by applying an optimal sequencing theorem for our multi-level tree network to obtain an optimal solution. In our rigorous simulation experiments, we considered a wide range of arbitrary graphs ranging from sparsely connected graphs to densely connected graphs with varying processor densities to observe the performance of all the proposed algorithms. The key concept of processor–link equivalence used in our algorithms elegantly builds up the "bridge" among sub-trees at different levels and thus makes it possible to schedule the load on an arbitrary graph.

From our experiments, we have shown the performance of our algorithms under several influencing factors such as, effect of network scalability, effect of network connectivity, and effect of load origination. As far as network scalability is concerned, our study clearly showed that in order to minimize the processing time, it may not be correct to bias our decision towards the choice of a larger network (in terms of number of processors), as eventually more P–L pairs may be eliminated from processing by our algorithms. This will typically happen especially when link speeds are comparable with processor speeds. On the other hand, with a small size network, with few P–L pairs getting eliminated, the performance may be better than the former case. Thus, merely scaling up the network size may not fetch a desirable performance. Secondly, our experiments highlighted the effect of network connectivity, a factor that corroborates the former findings. Thus, by increasing the degree of connectivity we can expect to minimize the processing time, since probability of more P–L pairs participating in the processing also increases. However, our experiments have also shown that this may not always guarantee a gain in performance. Finally, our study on the effect of load origination recommends the choice to submit the processing load on a network to achieve a minimum processing time.

Several extensions seem to be possible at this stage for the problem addressed in this paper. Firstly, since the primary motivation of the study in this paper is to develop algorithms and to evaluate their performance from theoretical standpoint, the implementation issues and possibly a real-life implementation on an application can be attempted to validate the theoretical findings reported in this paper. Some of the issues arising during implementation of DLT paradigm to real-life applications are discussed in [23,25]. Secondly, we have used a uniport communication model for all the strategies proposed in this paper. It would be interesting to design load distribution strategies for arbitrary graphs employing a multi-port model. Thirdly, although we can make use of any standard algorithms to generate G_{MST}, it should be noted that regardless of how G_{MST} is constructed from a given graph, scheduling algorithms proposed in this paper can always be employed to yield an optimal load distribution.

Furthermore, another possible concern on sequencing is that, during the load distribution process, instead of following a fixed sequence in every single-level tree network in G_{MST}, we may consider to allow the distribution sequences in different single-level tree networks to be diversified. That is to say, in the entire multi-level tree network, parent of each single-level tree network will follow its own particular sequence to disseminate the load while in this paper, in contrast to our approach using a fixed sequence (from the left hand side to the right hand side) in every single-level tree network. Lastly, it will be useful to design an optimal distribution strategy which can be applied on the graph networks directly without constructing a MST. Future extensions can also tackle the problem with some practical constraints and consider the case where multiple loads originating at different sites.

Appendix

As a first step, for the purpose of continuity, we state the implications of Rule A now and then present the procedure to obtain an equivalent reduced single-level tree. Consider a single-level tree $\Sigma(x, i, m+1)$ shown in figure 7(a). It has been shown in [7] that to obtain an optimal solution for a load distribution problem in the case of single-level tree networks, as a first step, we need to eliminate redundant P–L pairs. In order to eliminate such redundant P–L pairs (slow processor–link pairs) from participating in the load computation process, we apply the following condition, referred to as Rule A, in the literature [7].

$$z_{i,k}T_{cm} < z_{i,(k+1,\dots,k+r)}T_{cm} + w_{i,(k+1,\dots,k+r)}T_{cp}, \tag{A.1}$$

where $z_{i,(k+1,\dots,k+r)}$ is the equivalent link speed parameter of links $l_{i,k+1}$ to $l_{i,k+r}$ and $w_{i,(k+1,\dots,k+r)}$ is the equivalent processor speed parameter of processors $p_{i,k+1}$ to $p_{i,k+r}$ in $\Sigma(x, i, m+1)$, respectively, as shown in figure 7(a) and as defined in [7]. The implications of the above expression can be easily understood in its simplest form, for $r = 1$, which states the following,

$$z_{i,k}T_{cm} < z_{i,(k+1)}T_{cm} + w_{i,(k+1)}T_{cp}. \tag{A.2}$$

Note that if the above condition is violated, then the time taken by the front-end of $p_{x,i}$ to communicate a load fraction to $p_{i,k}$ through $l_{i,k}$ is more than the time taken to communicate the same load through $l_{i,k+1}$ and process it at $p_{i,k+1}$. Hence, it is logical to send the load to $p_{i,k+1}$ than to $p_{i,k}$. The above condition formalizes this logic for a set of processors that can share the load of a redundant P–L pair, thus eliminating it from the system. Thus, we systematically apply this Rule A, starting from the right hand side (P–L pair $(p_{i,(m-1)}, l_{i,(m-1)})$) and obtain an optimal reduced tree $\Sigma^*(x, i, n+1)$, where $n \leqslant m$, which is proven to be optimal in time performance in the literature. From [7], we obtain the expressions for $w_{i,(k+1,\dots,k+r)}$ and $z_{i,(k+1,\dots,k+r)}$ as

$$w_{i,(k+1,\dots,k+r)} = \left(\frac{1}{1+\sum_{u=k+2}^{k+r}\prod_{v=u}^{k+r} f_v}\right) w_{i,(k+r)}, \tag{A.3}$$

$$z_{i,(k+1,\dots,k+r)} = \frac{\sum_{u=k+2}^{k+r}\{(\prod_{v=u}^{k+r} f_v) z_{i,(u-1)}\} + z_{i,(k+r)}}{1+\sum_{u=k+2}^{k+r}\prod_{v=u}^{k+r} f_v}, \tag{A.4}$$

where f_v is as defined in (3). Using the above concept of equivalent processor and equivalent link to the entire single-level tree network, we can obtain the processing time for the entire single-level tree network as,

$$\begin{aligned} T\big(\Sigma(x,i,m+1)\big) &= \alpha_{x,i} w_{x,i} T_{cp} \\ &= \left(\frac{\prod_{v=1}^{m} f_v}{1+\sum_{u=1}^{m}\prod_{v=u}^{m} f_v}\right) w_{x,i} T_{cp} \\ &= 1 \cdot w_{x,\mathrm{eq}(i)} \cdot T_{cp}, \end{aligned} \tag{A.5}$$

where

$$w_{x,\mathrm{eq}(i)} = \left(\frac{\prod_{v=1}^{m} f_v}{1+\sum_{u=1}^{m}\prod_{v=u}^{m} f_v}\right) w_{x,i}. \tag{A.6}$$

Thus, given a single-level tree network, we can replace the entire network with a processor $p_{x,\mathrm{eq}(i)}$, as shown in figure 7(b), with its equivalent speed parameter given by (A.6). Further, we use this procedure in a recursive fashion, explained in section 3.1 to obtain G_{opt} from G_{MST}.

Thus, the optimal processing time is given by,

$$T\big(\Sigma(x,i,m+1)\big) = \alpha_{x,\mathrm{eq}(i)} \cdot L \cdot w_{x,\mathrm{eq}(i)} \cdot T_{cp} \tag{A.7}$$

and for $\Sigma(0, m+1)$, the optimal processing time is given by

$$\begin{aligned} T\big(\Sigma(0,m+1)\big) &= \alpha_{\mathrm{eq}(0)} \cdot L \cdot w_{\mathrm{eq}(0)} \cdot T_{cp} \\ &= L \cdot w_{\mathrm{eq}(0)} \cdot T_{cp}, \end{aligned} \tag{A.8}$$

which is equal to the processing time of load L. It may be noted that, (A.8) is indeed the total optimal processing time of the entire load on $G_{\mathrm{opt}}/G_{\mathrm{opt}}^{\mathrm{os}}$.

References

[1] M.M. Eshaghian (ed.), *Heterogeneous Computing* (Artech House, 1996).

[2] K. Hwang and Z. Xu, *Scalable Parallel Computing: Technology, Architecture, Programming* (McGraw-Hill, New York, 1998).

[3] L. Pangfeng and N.B. Sandeep, Experiences with parallel N-body simulation, IEEE Transactions on Parallel and Distributed Systems 11(12) (2000) 1306–1323.

[4] K.Y. Tieng, F. Ophir and L.M. Robert, Parallel computation in biological sequence analysis, IEEE Transactions on Parallel and Distributed Systems 9(3) (1998) 283–294.

[5] B.A. Shirazi, A.R. Hurson and K.M. Kavi, *Scheduling and Load Balancing in Parallel and Distributed Systems* (IEEE Computer Society Press, Los Alamitos, CA, 1995).

[6] Y.C. Cheng and T.G. Robertazzi, Distributed computation with communication delays, IEEE Transactions on Aerospace and Electronic Systems 24 (1998) 700–712.

[7] V. Bharadwaj, D. Ghose, V. Mani and T.G. Robertazzi, *Scheduling Divisible Loads in Parallel and Distributed Systems* (IEEE Computer Society Press, Los Alamitos, CA, 1996).

[8] M. Drozdowski, *Selected Problems of Scheduling Tasks in Multiprocessor Computer Systems*, Series Monographs, No. 321 (Poznan University of Technology Press, Poznan, Poland, 1997).

[9] V. Bharadwaj, D. Ghose and T.G. Robertazzi, Divisible load theory: A new paradigm for load scheduling in distributed systems, Cluster Computing, Special Issue on Divisible Load Scheduling (January 2003).

[10] J. Sohn and T.G. Robertazzi, Optimal divisible job load sharing on bus networks, IEEE Transactions on Aerospace and Electronic Systems 1 (1996).

[11] J. Blazewicz, M. Drozdowski and M. Markiewicz, Divisible task scheduling – concept and verification, Parallel Computing 25 (January 1999) 87–98.

[12] G. Barlas, Collection-aware optimum sequencing of operations and closed-form solutions for the distribution of a divisible load on arbitrary processor trees, IEEE Transactions on Parallel and Distributed Systems 9(5) (1998) 429–441.

[13] T.H. Jui, H.J. Kim and T.G. Robertazzi, Scalable scheduling in parallel processors, in: *Proceedings of Conference on Information Sciences and Systems*, Princeton University (March 20–22, 2002).

[14] J. Blazewicz, M. Drozdowski, F. Guinand and D. Trystram, Scheduling a divisible task in a 2-dimensional mesh, Discrete Applied Mathematics 94(1–3) (June 1999) 35–50.

[15] D. Ghose and H.J. Kim, Load partitioning and trade-off study for large matrix-vector computations in multicast bus networks with communication delays, Journal of Parallel and Distributed Computing 54 (1998).

[16] K. Li, Managing divisible loads in partitionable networks, in: *High Performance Computing Systems and Applications*, eds. J. Schaeffer and R. Unrau (Kluwer Academic, Dordrecht, 1998) pp. 217–228.

[17] D.A.L. Piriyakumar and C.S.R. Murthy, Distributed computation for a hypercube network of sensor-driven processors with communication delays including setup time, IEEE Transactions on Systems, Man and Cybernetics – Part A: Systems and Humans 28(2) (March 1998) pp. 245–251.

[18] V. Bharadwaj and G. Barlas, Efficient scheduling strategies for processing multiple divisible loads on bus networks, Journal of Parallel and Distributed Computing 62 (2002) 132–151.

[19] V. Bharadwaj, H.F. Li and T. Radhakrishnan, Scheduling divisible loads in bus networks with arbitrary processor release times, Computer Math. Appl. 32(7) (1996).

[20] J. Blazewicz and M. Drozdowski, Distributed processing of divisible jobs with communication startup costs, Discrete Applied Mathematics 76(1–3) (June 1997) 21–41.

[21] V. Bharadwaj, X.L. Li and C.C. Ko, On the influence of start-up costs in scheduling divisible loads on bus networks, IEEE Transactions on Parallel and Distributed Systems 11(12) (2000) 1288–1305.

[22] X. Li, V. Bharadwaj and C.C. Ko, Divisible load scheduling on single-level tree networks with finite-size buffers, IEEE Transactions on Aerospace and Electronic Systems 36(4) (October 2000).

[23] V. Bharadwaj and S. Ranganath, Theoretical and experimental study on large size image processing applications using divisible load paradigm on distributed bus networks, Image and Vision Computing 20 (2002) 917–935.

[24] M. Drozdowski and P. Wolniewicz, Experiments with scheduling divisible tasks in clusters of workstations, *Euro-Par 2000*, Lecture Notes in Computer Science, Vol. 1900 (Springer, Berlin, 2000) pp. 311–319.

[25] S.K. Chan, V. Bharadwaj and D. Ghose, Experimental study on large size matrix-vector product computations using divisible load paradigm on distributed bus networks, #TR-OSSL/BV-DLT/01 (November 2000).

[26] V. Bharadwaj and G. Barlas, Access time minimization for distributed multimedia applications, Multimedia Tools and Applications 2/3 (November 2000) 235–256.

[27] H. Wen-Jing, J.C. Moon and D. Amitabha, Linear recursive networks and their applications in distributed systems, IEEE Transactions on Parallel and Distributed Systems 8(7) (1997) 673–680.

[28] T.H. Cormen, C.E. Leiserson and R.L. Rivest, *Introduction to Algorithms* (The MIT Press, Cambridge, MA, 1994).

[29] Y.C. Cheng and T.G. Robertazzi, Distributed computation for a tree network with communication delays, IEEE Transactions on Aerospace and Electronic Systems 26(3) (1990) 511–516.

Jingman Yao
E-mail: engp0969@nus.edu.sg

Bharadwaj Veeravalli
E-mail: elebv@nus.edu.sg