Available online at www.sciencedirect.com
ScienceDirect
J. Parallel Distrib. Comput. 66 (2006) 1404–1418
Journal of Parallel and Distributed Computing
www.elsevier.com/locate/jpdc

Distributed scheduling strategy for divisible loads on arbitrarily configured distributed networks using load balancing via virtual routing

Zeng Zeng, Bharadwaj Veeravalli*

Department of Electrical and Computer Engineering, The National University of Singapore, Singapore 117576, Singapore

Received 30 May 2005; received in revised form 20 April 2006; accepted 16 May 2006
Available online 13 July 2006

Abstract

In this paper, we consider a scheduling problem for divisible loads originating from single or multiple sites on arbitrary networks. We first propose a generalized mathematical model and formulate the scheduling problem as an optimization problem with an objective to minimize the processing time of the loads. We derive a number of theoretical results on the solution of the optimization problem. On the basis of these first set of results, we propose an efficient algorithm for scheduling divisible loads using the concept of load balancing via *virtual routing* for an arbitrary network configuration. The proposed algorithm has three major attractive features. Firstly, the algorithm is simple to realize and can be implemented in a *distributed* fashion. The second one is in its style of working by avoiding the need for generating a timing diagram explicitly for any complex networks having an arbitrary network topology. The last one is its capability of handling divisible loads originating from both single and multiple sites. When divisible loads originate from a single node, we compare the proposed algorithm with a recently proposed RAOLD algorithm which is based on minimum cost spanning tree [J. Yao, V. Bharadwaj, Design and performance analysis of divisible load scheduling strategies on arbitrary graphs, Cluster Computing 7(2) (2004) 191–207]. When divisible loads originate from multiple sites, we test the performance on sparse, medium and densely connected networks. This is the first time in the divisible load theory (DLT) literature that such a generic approach for handling divisible loads originating from multiple sites on arbitrary networks employing load balancing via virtual routing is attempted.

Keywords: Scheduling; Arbitrary networks; Minimum cost spanning tree; Routing path; Divisible load theory; Processing time; Load balancing

1. Introduction

Parallel and distributed heterogeneous computing has been proven to be an efficient and successful way for several real-life applications [16,11]. One of the primary issues in the area of parallel and distributed computing is on how to partition and schedule loads that arrive at the processing nodes among the available system resources so that the best performance is achieved with respect to finish time of all input loads. To efficiently utilize the system resources, researchers have proposed a number of load/task scheduling and balancing strategies in the literature [27,22,28].

Based on the types of loads under processing, load scheduling/balancing problem can be classified into two categories: indivisible load scheduling and divisible load scheduling. Indivisible loads are atomic and cannot be divided into smaller sub-tasks, and have to be processed in its entirety on a processor. Divisible loads are data parallel loads that are arbitrarily partitionable amongst nodes of the network. In contrast to the indivisible loads model, divisible loads are assumed to be very large in size, homogeneous, and arbitrarily divisible in the sense that, each partitioned portion of the loads can be independently processed on any processor in the system [6,7]. The theory of scheduling and processing of divisible loads, referred to as *Divisible Load Theory* (DLT), has stimulated considerable interest among researchers in the field of parallel and distributed systems since its origin in 1988 [12,1]. The primary concern in the research of DLT is to determine the optimal fractions of the entire loads to be assigned to each of the processors in such a way that the total processing time of the entire loads is a minimum. Compilation of all the research contributions in DLT until 1995 can be found in monographs

* Corresponding author. Fax: +65 777 8804.
E-mail addresses: elezz@nus.edu.sg (Z. Zeng), elebv@nus.edu.sg (B. Veeravalli).

0743-7315/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2006.05.007

[6,16]. Two recent survey articles [7,23] consolidate all the results until 2002.

Numerous studies have been conducted in the DLT literature and 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 processing must stop at the same time instance. This condition is referred to as an *optimality principle* in the DLT literature and analytic proof can be found in [24,6,10]. In [19], 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, in which all the incident links on a processor are concurrently used. The advantage of the multi-port model was also demonstrated in a two-dimensional mesh network. In [11], load partitioning of intensive computations of large matrix–vector products in a multi–cast bus network was investigated.

To determine the ultimate speedup using DLT analysis, Li [20] conducted an asymptotic analysis for partitionable network topologies. Most recent studies focus on system-dependent constraints such as, scheduling under finite buffer capacity constraints [5], estimating the processor and link speeds by some methods of probing [17], scheduling divisible loads on Mesh multiprocessor architectures [18], to quote a few. Also, the applicability of DLT concepts to schedule and process loads generated from large-scale physics experiments (RHIC at BNL, USA) on computational grids are investigated in [26]. Finally, use of affine delay models for communication and computation components for scheduling divisible loads are extensively studied in [15,8], etc.

All the above studies focused their attention in designing load distribution strategies under the assumption that all loads originate at one processor. For the solution approaches, the algorithms proposed in the literature were centralized. In this paper, we will consider a distributed load scheduling problem with divisible loads originating from single or multiple processors on arbitrary networks. This scenario happens commonly in realistic situations, such as applications in distributed real-time systems.

1.1. Our contributions

Our contributions are summarized as follows. We formulate the scheduling problem with divisible loads originating from single or multiple sites on arbitrary networks as a real-valued constrained optimization problem. On the basis of load balancing via virtual routing proposed in [28], we design a distributed scheduling strategy to achieve the optimal processing time of all loads in the system. In our proposed algorithm, each processor can determine the amount of loads that should be transferred to other processors and the amount of loads that should be processed locally, according to some local information. This is the first time that a *distributed algorithm* is attempted and proposed in the DLT literature. In all the earlier DLT literature, timing diagrams were used to precisely define the load distribution process. This timing diagram representation would be meaningful if one could easily conceive of strategies that address scheduling loads from a single site and the underlying topology. However, when one needs to schedule multiple divisible loads originating from several sites on an arbitrary topology, it is rather impossible to capture the load distribution process by a single timing diagram. The main difficulty that lies in this approach would be to explicitly schedule several timing components such as computation and communication from one site to other sites, identifying which processor-link pairs are redundant [2,6], etc. Thus, in this paper, we take a radically different approach to address this complex problem by carefully formulating as a generalized minimization problem, thus avoiding the need for a timing diagram representation.

We derive a number of theoretical results on the solution of the optimization problem. In case divisible loads originate from a single site, we compare our proposed algorithm with a recently proposed RAOLD algorithm [9]. It is demonstrated that the proposed algorithm performs better than RAOLD in terms of the processing time. We analyze the difference between the divisible loads originating from single and multiple sites in our proposed algorithm. When divisible loads originate from multiple sites on arbitrary networks, we prove that our proposed algorithm also can solve the scheduling problem efficiently by numerous simulation experiments.

The organization of the paper is as follows. In Section 2, we present a mathematical model, the problem definition and formulation. In Section 3, we propose our strategy and derive conditions for obtaining an optimal solution. In Section 4, we present a detailed illustrative example to show the complete workings of our proposed algorithm for ease of understanding. We shall also present numerical studies to quantify the performance in the cases that when divisible loads originate from single and multiple sites. We highlight our contributions and discuss on possible future extensions in Section 5.

2. Mathematical model and problem formulation

In this section, we shall formally introduce the problem we are tackling and propose a mathematical model used in our analysis. We will also introduce the required terminology, notations and definitions that will be used throughout this paper. In the following, we shall first describe the characteristics of the system, the resource constraints and some assumptions that are considered in the problem context.

2.1. Description of system, assumptions and notations

We consider the problem of scheduling and processing divisible loads on an arbitrary computer network system [14] as shown in Fig. 1(a). The system consists of n heterogeneous nodes, which represent host computers, interconnected by a generally configured communication/intercommunication network. Further, these nodes may differ in configurations such as, speed characteristics and number of resources. However, we assume that they have the same processing capabilities. For instance, a divisible load can be processed at any node in the

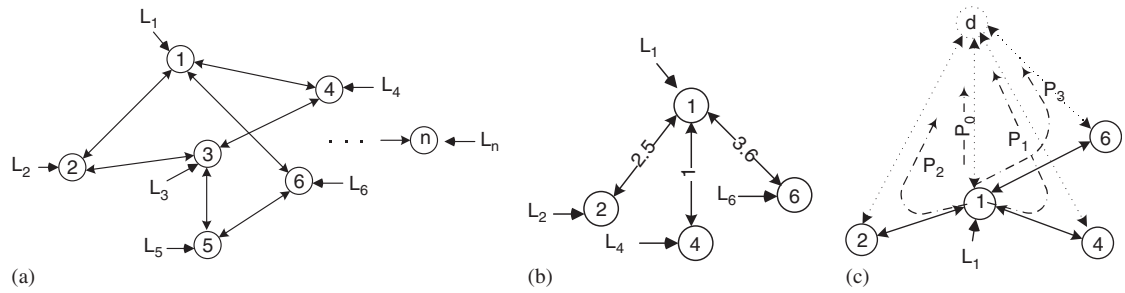

Fig. 1. (a) An arbitrary computer network system with multiple loads; (b) example 1 of single-level tree; (c) single-level tree with virtual node and virtual links.

system. Here, we use N to denote the set of nodes, i.e., $n = |N|$ and U to denote a set whose elements are unordered pairs of distinct elements of N. Each unordered pair $e = \langle i, j \rangle$ in U is called an *edge*. For each edge $\langle i, j \rangle$, we define two ordered pairs (i, j) and (j, i) which are called *links* and we denote W as the set of links. A node i is said to be a *neighboring node* of j, if i is connected to j by an edge. For a node j, let $V_j = \{i, |(i, j) \in W\}$ denote a set of neighboring nodes of node j.

We assume that all nodes in the system have front-end [7]. Each node can take care of transferring the load to its neighboring nodes one at a time. This assumption is reasonable because the buffer size at a node is limited and it will not have enough loads to send to all of its neighboring nodes simultaneously. We also assume that a node can receive loads transferred from multiple sites simultaneously. We further assume that node i cannot start processing until it receives all the loads. This is due to the fact that when the node is receiving the loads, it does not know which part of the loads should be processed locally and which part should be transferred to which neighboring nodes [27,6,26]. When node i receives all the loads, it can start sending the loads to its neighboring nodes one by one and processing at the same time.

We consider divisible loads originating from single or multiple nodes in the network and denote L_i as the amount of loads originating at node i for processing, where $L_i \geqslant 0$ holds. Let l_i denote the amount of loads assigned to node i according to a scheduling strategy and x_{ij} denote the amount of loads which should be transferred from node i to node j, $j \in V_i$. Hence, the following conservation equations hold:

$$\gamma_i = L_i + \sum_{j \in V_i} x_{ji} = l_i + \sum_{j \in V_i} x_{ij}, \tag{1}$$

where γ_i is the total amount of loads at node i. In general, there is a node delay incurred when l_i load is processed at node i and a communication delay incurred when x_{ij} load is transferred on link (i, j). In DLT, the above two kinds of delay functions are assumed to be linear, increasing functions [27,6,9,7]. We shall denote $P_i(l_i) = E_i \times l_i$ as the node delay function for node i and $G_{ij}(x_{ij}) = C_{ij} \times x_{ij}$ as the communication delay function on link (i, j), where, E_i is the time it takes to compute a unit load by node i and C_{ij} is the time it takes to transfer a unit load on link (i, j), respectively. We adopt a simple linear model, rather than an affine cost model involving start-ups, both for computation and communication, because only large-scale applications are likely to be deployed on heterogeneous platforms and each step of the procedure will be both computation and communication intensive, so that the start-up overheads can indeed be neglected.

In this paper, our objective is to obtain an optimal load distribution such that the processing time of the entire loads is a minimum. For each node i, the processing time T_i is dependent on l_i and $t_i^{\max}$, where $t_i^{\max}$ is the time instance at which the last load is received by node i. Obviously, if no load is sent to node i by its neighboring nodes, $t_i^{\max} = 0$ holds. Now, we can obtain:

$$T_i = t_i^{\max} + E_i \times l_i. \tag{2}$$

We denote $t_i^{\max}(j)$ as the time instance at which the last load coming from node j is received by node i. According to the definition of $t_i^{\max}$, we obtain the following equation:

$$t_i^{\max} = \max\{t_i^{\max}(j), |\forall j \in V_i\}. \tag{3}$$

We can observe that $t_i^{\max}$ may be determined by node j. Let $t_{i-j}^{\max}$ denote the time instance at which the last load coming from V_i, *except* node j, is received by node i.

As we mentioned before, when node i receives all the loads, it can start processing and transferring some loads to one of its neighboring nodes simultaneously according to the scheduling. In the DLT literature [27,6], it has been shown that the processing time can be improved if we follow a sequence in which link speeds decrease (referred to as, optimal sequence). Thus, we can observe that it is important for node i to obtain such an optimal sequence, following which node i should send loads to its neighboring nodes one by one. We will discuss the optimal sequence in Section 3 in detail. Now, we use S_i to re-index the nodes of V_i, the global number of the neighboring nodes, as $S_i = \{1, 2, \ldots, v_i\}$, $v_i = |V_i|$, for ease of mathematical representation. Thus, we can conceive an inverse function that maps the nodes of S_i to V_i and we denote such an inverse function of $S_i(j)$ as $S_i^{-1}(k)$. Thus, $S_i(j) = k$ and $S_i^{-1}(k) = j$ imply $k \in S_i$ and $j \in V_i$, respectively. For example, referring to Fig. 1(a), node 1 has three neighboring nodes and $V_1 = \{2, 4, 6\}$. For any sequence of node 1, $S_1 = \{1, 2, 3\}$. If node 1 sends load to node 2 first, then node 6, and sends load to node 4 last. We can obtain that $S_1^{-1}(1) = 2$, $S_1^{-1}(2) = 6$, $S_1^{-1}(3) = 4$. If we know node 2 is in V_1, we also can obtain the position of node 2 in S_1 by $S_1(2) = 1$. Hence, if node j, $j \in V_i$, sends

loads to node i, the following equations hold:

$$t_i^{\max}(j) = t_j^{\max} + \sum_{k=1}^{S_j(i)} C_{jS_j^{-1}(k)} \times x_{jS_j^{-1}(k)}. \quad (4)$$

From (3), (4) and from the definition of $t_{i-j}^{\max}$, we obtain:

$$t_i^{\max} = \max\{t_i^{\max}(j), t_{i-j}^{\max}\} = \max\left\{ t_j^{\max} + \sum_{k=1}^{S_j(i)} C_{jS_j^{-1}(k)} \times x_{jS_j^{-1}(k)}, |\forall j \in V_i, x_{ji} > 0 \right\}.$$

Note that the processing time of the system is influenced by l_i and the transfer amount x_{ij}. Now we shall present a formal definition of the problem addressed in this paper.

2.2. Problem formulation

Based on the model introduced in Section 2.1, we shall formally define the problem that we want to address. In essence, we formulate the problem as a real-valued optimization problem with the objective of minimizing the processing time of the system according to (2). We state the following:

$$\text{Minimize}: \quad D(l, x) = \max\{T_i = t_i^{\max} + E_i \times l_i, |\forall i \in N\}, \quad (5)$$

where

$$t_i^{\max} = \max\left\{ t_j^{\max} + \sum_{k=1}^{S_j(i)} C_{jS_j^{-1}(k)} \times x_{jS_j^{-1}(k)}, |\forall j \in V_i, x_{ji} > 0 \right\},$$
$$l = [l_1, l_2, \ldots, l_n], x = [x_{ij}], (i, j) \in W$$

$$\text{subject to:} \quad L_i + \sum_{j \in V_i} x_{ji} = l_i + \sum_{j \in V_i} x_{ij}, \quad (6)$$

$$\sum_{i \in N} L_i = \sum_{i \in N} l_i, \quad (7)$$

$$l_i \geqslant 0, i \in N, \quad (8)$$

$$x_{ij} \geqslant 0, (i, j) \in W. \quad (9)$$

Eq. (6) defines load balancing constraints, each of which equates the amount of loads in and out of node i. In order to obtain an optimal solution to the problem defined by (5), each node i can determine l_i and x_{ij}, $j \in V_i$, according to the total amount of load γ_i. For node i, another important parameter $t_i^{\max}$ (independent of node i) is only determined by the incoming amount of loads x_{ji}. Hence, the load scheduling strategy should determine the values of l, x, and the optimal sequence of each node sending loads to its neighboring nodes.

We shall now introduce the main idea we used to attack the problem. In [28], the authors proposed a distributed Load Balancing algorithm via virtual routing. In [21], the divisible load scheduling problem on single-level tree topology was analyzed. Note that, if we use a distributed method to solve the problem addressed in this paper, each node will only consider itself and its neighboring nodes. If node i has loads to send to its neighboring nodes, we can obtain a reduced single-level tree (star) graph with node i as the root processor. For example, referring to Fig. 1(a), if node 1 has loads to send to node 2, 4, 6, we can obtain a sub-graph as shown in Fig. 1(b). Then, node 1 can calculate l_1, and x_{12}, x_{14}, x_{16} by virtual routing method proposed in [28], which has been proven to be an efficient distributed search method. All the nodes in the system construct their own single-level tree and then calculate the l and x according to the optimal sequences one by one. After some iterations, an optimal solution can be found, if it exists. We will discuss the optimal solution in detail in the following section.

3. Proposed strategy for optimal solution

In this section, we will propose our solution to approach the problem defined in (5)–(9). As we discussed in Section 2, we shall first present the sub-algorithm for each node to search for the optimal solution in each iteration and then discuss the optimal sequence, based on which we construct our scheduling strategy. After that, we will present the scheduling strategy for divisible loads originating from a single site first and discuss the difference between divisible loads originating from single and multiple sites, for ease of simplicity. After that, we will present the scheduling strategy for divisible loads originating from multiple sites. At the end of this section, we will prove the convergence of our strategy and deduce the complexities of the proposed algorithm.

3.1. Sub-algorithm for a node and optimal sequence

As proved in [27,6], a set of l and x is an optimal solution to problem (5) *if and only if* all the nodes in the system stop processing at the same time. For our distributed scheduling strategy, each node considers its neighboring nodes and constructs a star sub-graph. Each node will try to obtain a local optimal solution among the sub-graph. As a first step, we present our sub-algorithm of a node to make all the nodes in the sub-graph stop processing at the same time.

Our sub-algorithm of a node is based on the virtual routing method proposed in [28]. In our sub-algorithm, each node i in the system will attempt to obtain a local optimal solution to make node i and node j, $j \in V_i$, to stop processing at the same time. At first, we assume that $L_i > 0$ and $L_j = 0$, and besides node i, no other node will send load to node j. Now we obtain the sub-problem of problem (5) for an optimal sequence $\bar{S}_i$ as follows:

$$\text{Minimize}: \quad D_i(l_i, x_i) = \max\{T_i, T_j, |\forall j \in V_i\}, \quad (10)$$

where

$$T_i = E_i \times l_i,$$
$$T_j = \sum_{k=1}^{S_i(j)} C_{i\bar{S}_i^{-1}(k)} \times x_{i\bar{S}_i^{-1}(k)} + E_j \times l_j, \forall j \in V_i, x_{i\bar{S}_i^{-1}(k)} > 0$$
$$\text{subject to:} \quad L_i = l_i + \sum_{j \in V_i} l_j \quad \text{with (6), (8), (9).}$$

The sub-algorithm will search for the solution of $l_i, l_j\ (= x_{ij})$ to make all the nodes among the sub-graph stop processing at the same time. In the rest of this paper, we use a bar over a symbol to indicate that this symbol is an optimal solution. Now we add a *virtual* node, which is referred to as the *destination* node (node d), into our network system. Further, we connect node d with node i and j, $j \in V_i$, by direct *virtual* links (i, d) and (j, d), respectively. We use functions $G_{id}(x_{id}) = E_i \times x_{id}$ and $G_{jd}(x_{jd}) = E_j \times l_j$ as the delay functions of virtual links (i, d) and (j, d), respectively. Note that node i has a set of routing paths P_i, through which node i can send loads to node d. Let p_i^0 denote the routing path of $i \to d$ directly and p_i^k denote the routing path of $i \to j \to d$, where $j = \bar{S}_i^{-1}(k)$. We denote x_i^k as the amount of loads that is transferred on path p_i^k and denote d_i^k as the time delay on path p_i^k. Obviously, $x_i^0 = l_i$, $x_i^k = x_{ij}$, and $d_i^0 = T_i$, $d_i^k = T_j$ hold, where $k = 1, 2, \ldots, v_i$, $j = \bar{S}_i^{-1}(k)$. After this transformation, we can describe the process of scheduling problem in another way. According to the *optimality principle* in the DLT literature, in order to obtain an optimal solution to the sub-problem (10), node i shall determine the amount of loads which is transferred on each path $p_i^k \in P_i$ and make the time delays of all the routing paths to be the same [27,6], which is:

$$\bar{d}_i^0 = \bar{d}_i^1 = \cdots = \bar{d}_i^{v_i}. \tag{11}$$

Let $d_i^{\min}$ denote the minimum delay routing path (MDRP) of node i, which is $\min\{d_i^k, |k = 0, 1, \ldots, v_i\}$. Clearly, an optimal solution of problem (10) results only if an amount of loads travel along MDRP paths in P_i. The amount of loads is strictly suboptimal *only if* there is a positive amount of loads that travels on a non-MDRP path. This suggests that a suboptimal solution can be improved by transferring some amount of loads to an MDRP path from other paths in P_i (evident from (11) above). However, by transferring all the loads to the MDRP path may lead to an oscillatory behavior. Thus, it would be better to transfer only part of the loads from other paths via MDRP path. Below, we shall determine the amount of loads from each of the non-MDRP paths in P_i to be transferred via MDRP to seek an optimal solution. From [4,28], we obtain that the iteration takes the form:

$$x_i^{k(r+1)} = \max\left\{0, x_i^{k(r)} - \alpha^{(r)} \frac{d_i^{k(r)} - d_i^{\min(r)}}{H_i^{k(r)}}\right\}, \quad k = 0, \ldots, v_i, \quad k \neq \min, \tag{12}$$

$$x_i^{\min(r+1)} = \gamma_i^{(r+1)} - \sum_{k=0, k\neq\min}^{v_i} (x_i^{k(r+1)}), \tag{13}$$

where, $\alpha^{(r)}$ is a positive scalar step-size in rth iteration, according to some rule [3], and we set $\alpha^{(r)}$ to 0.5 in each iteration, which works well in our simulations, and min presents the MDRP path in set $d_i^{(r)}$ and

$$H_i^{k(r)} = \frac{\partial d_i^{k(r)}}{\partial x_i^{k(r)}} + \frac{\partial d_i^{\min(r)}}{\partial x_i^{\min(r)}}. \tag{14}$$

For ease of understanding, we consider Example 1 as shown in Fig. 1(b), in which $L_1 = 100$ units loads and $L_2 = L_4 = L_6 = 0$ and node 1 tries to send some loads to its neighboring nodes. The processor speed parameters are $E_1 = 2.0$, $E_2 = 4.0$, $E_4 = 1.0$, $E_6 = 1.5$ and the respective link speed parameters are marked in the graph. At first we choose a random sequence for node 1, as sending loads to node 2 first, and then node 4, 6, one by one. We obtain $S_1(2) = 1$, $S_1(4) = 2$ and $S_1(6) = 3$. After adding the virtual destination node d and the virtual links $(1, d)$ and (j, d), $j \in V_1$, we obtain Fig. 1(c), where P indicates the routing paths of node 1. In order to obtain (11), we first set an initial feasible solution that is $x_1^{(0)} = [x_1^{0(0)}, x_1^{1(0)}, x_1^{2(0)}, x_1^{3(0)}] = [100, 0, 0, 0]$, $r = 0$ firstly. We can obtain $d_1^0 = [100 \times 2, 0, 0, 0] = [200, 0, 0, 0]$, where we can choose P_2 as the MDRP, and then from (14) we can obtain $H_1^0 = [8.5, -, 8.5, 11.6]$. By using (12) and (13), we obtain $x_1^{(1)} = [88.24, 11.76, 0, 0]$ in the first iteration. After 11 iterations, we can observe that the time delay on each path becomes the same that is $\bar{d}_1^0 = \bar{d}_1^1 = \bar{d}_1^2 = \bar{d}_1^3 = 97.86$ s. The solution for this example is $l_1 = 48.93$, $x_{12} = 15.06$, $x_{14} = 30.11$ and $x_{16} = 5.90$ units loads.

By adding a virtual node d and virtual links into the system, we transfer the divisible load scheduling problem into an equivalent routing problem, in which each node in the system constructs its own routing paths and decides the amount of loads that should be transferred on each path. By this way, we consider the computation delay and communication delay as the link delays on the path, and then we can solve the optimal virtual routing problem by using some search methods, such as Golden Section Search [25], Newton's method [28], to avoid formulating the complicated closed-form equations which may even not exist in some cases, e.g., NP-complete problems [22]. Now we state the following theorem.

Theorem 1 (*Optimal sequence*). *In a single-level tree network, in order to achieve minimum processing time, the sequence of load distribution by the root node i should follow the order in which the link speeds* $(\frac{1}{C_{ij}},\ j \in V_i)$ *decrease* [21,6].

Theorem 1 and Eq. (11) together provide a necessary and sufficient condition for solving problem (10). According to this condition, the loads should be distributed first through the fastest link, then through the second fastest link, and so on until the slowest link is assigned the last load fraction and all the nodes should stop processing loads at the same time. Thus, by adopting such a sequence of load distribution according to the speeds of the links and finding a solution to make all the nodes stop processing at the same time, an optimal processing time will be achieved.

Thus, we have provided a *necessary and sufficient* condition for the optimal solution of the sub-problem (10) and iteration functions (12) and (13) which act as a building block of the proposed scheduling strategy. In the following section, we will present our scheduling strategy for divisible load originating from single and multiple sites on arbitrary networks.

3.2. Scheduling strategy for an arbitrary topology

In this section, using the sub-algorithm for each node in the system described above, we shall derive our scheduling strategy systematically. Though the basic idea of the solutions for the scheduling problem of divisible load originating from single and multiple sites is the same, there are some difference between the two cases. Hence, we will present the strategy for loads originating from a single site first and then discuss the main difference of the two cases. Lastly, we will propose the strategy for loads originating from multiple sites on arbitrary networks. In the following, we should first state an important lemma for an arbitrary topology.

Lemma 1. *For a node i in the system, in case* $t_j^{\max} \geqslant t_j^{\max}(i)$, $j \in V_i$, *the optimal sequence in Theorem* 1 *for node i still holds.*

Proof. From Theorem 1, we can observe that for node i, the optimal sequence is independent of l_j, the amount of loads to be processed at node j, and E_j, the processing capability of node j, where $j \in V_i$. For a given l_j, the time instance at which node j starts processing may be later than the time when node j receives the load coming from node i. If we increase the processing speed ($\frac{1}{E_j}$) to a larger value, we can guarantee that node j can finish computing on l_j in time. Hence, we can draw a conclusion that the optimal sequence of node i is also independent of $t_j^{\max}$, the time instance at which node j receives all the loads from its neighboring nodes. □

Theorem 1 is only applied in single-level tree network, whereas Lemma 1 guarantees that when node i constructs its own single-level tree sub-graph in the network, the optimal sequence in Theorem 1 still holds and the optimal sequence is independent of the status of node i and its neighboring nodes. Now, we shall make a small modification of sub-problem (10) as following:

$$\text{Minimize}: \quad D_i(l_i, x_i) = \max\{T_i, T_j, |\forall j \in V_i\}, \tag{15}$$

where

$$T_i = t_i^{\max} + E_i \times l_i, \quad T_j = t_j^{\max} + E_j \times l_j,$$

with the same constraints in (5). According to Lemma 1, we can observe that the optimal sequence still holds. Hence, (12) and (13) can also be used to seek an optimal solution for the sub-problem (15).

3.2.1. Divisible load originating from a single site

We consider Fig. 1(a) to discuss the scheduling strategy for divisible loads originating from a single site. In this example, we assume $L_1 = 100$ and $L_2 = L_3 = \cdots = L_n = 0$. Similar to Fig. 1(c), by adding a virtual node d and virtual links (i, d) and (j, d), we can obtain a set of routing paths of node i in Fig. 1(a). The optimal sequence of each node i, $\bar{S}_i$, can be predefined before the calculation. Each node i in the system will try to find an optimal solution for the single-level tree sub-graph, the root processor of which is node i itself. In each iteration, the nodes calculate one by one, till all the nodes have finished calculating. In rth iteration, node i has its own total amount of loads $\gamma_i^{(r)} = L_i + \sum_{j \in V_i} x_{ji}^{(r)}$, $(L_i = 0, i \neq 1)$. In order to avoid job shuttles within the system, node i will not calculate load amount to be transferred on the link (i, j), if $x_{ji}^{(r)} > 0$. Node i calculates $x_i^{k(r)}$ on each routing path one by one and skips the path $(i \to j \to d)$ if $x_{ji}^{(r)} > 0$. Obviously, $t_{j-i}^{\max(r)}$, $t_i^{\max(r)}$, and $x_{ji}^{(r)}$, $x_{mn}^{(r)}$, for $m \neq i, n \neq i, (m, n) \in W$, are independent of node i in rth iteration. Before node i starts calculating $l_i^{(r+1)}$ and $x_{ij}^{(r+1)}$, $(x_{ji}^{(r)} = 0)$, node i can obtain $t_i^{\max(r)}$ and $t_{j-i}^{\max(r)}$ by the following equations:

$$t_i^{\max(r)} = \max\left\{ t_j^{\max(r)} + \sum_{k=1}^{\bar{S}_j(i)} C_{j\bar{S}_j^{-1}(k)} \times x_{j\bar{S}_j^{-1}(k)}^{(r)}, |\forall j \in V_i, x_{ji}^{(r)} > 0 \right\},$$

$$t_{j-i}^{\max(r)} = \max\left\{ t_n^{\max(r)} + \sum_{k=1}^{\bar{S}_n(j)} C_{n\bar{S}_n^{-1}(k)} \times x_{n\bar{S}_n^{-1}(k)}^{(r)}, |\forall n \in V_j, n \neq i, x_{nj}^{(r)} > 0 \right\}.$$

However, node i can determine $t_j^{\max(r)}(i)$ as

$$t_j^{\max(r)}(i) = t_i^{\max(r)} + \sum_{k=1}^{\bar{S}_i(j)} C_{i\bar{S}_i^{-1}(k)} \times x_{i\bar{S}_i^{-1}(k)}^{(r)} \quad j \in V_i$$

by the transfer vector $x_i^{(r)}$ in rth iteration. Thus, we can obtain the time instance at which node j finishes receiving loads from its neighboring nodes, which is

$$t_j^{\max(r)} = \max\{t_j^{\max(r)}(i), t_{j-i}^{\max(r)}\}. \tag{16}$$

From (16), we can obtain the time delay on each path in P_i in rth iteration:

$$\begin{aligned} d_i^{0(r)} &= t_i^{\max(r)} + E_i \times x_i^{0(r)}, \\ d_i^{k(r)} &= t_j^{\max(r)} + E_j \times l_j^{(r)}, \\ &\qquad k = 1, 2, \ldots, v_i, \quad j = \bar{S}_i^{-1}(k), \end{aligned} \tag{17}$$

where, $l_j^{(r)} = \gamma_j^{(r)} - \sum_{m \in V_j} x_{jm}^{(r)}$. We can also obtain the first derivative of the time delay on each path for node i as shown in the following:

$$\frac{\partial d_i^{0(r)}}{\partial x_i^{0(r)}} = E_i,$$

$$\frac{\partial d_i^{k(r)}}{\partial x_i^{k(r)}} = \begin{cases} E_j & \text{if } t_j^{\max(r)}(i) < t_{j-i}^{\max(r)}; \\ E_j + C_{ij} & \text{if } t_j^{\max(r)}(i) \geqslant t_{j-i}^{\max(r)}, \end{cases} \tag{18}$$

where, $k = 1, 2, \ldots, v_i$, $j = \bar{S}_i^{-1}(k)$. Thus, we have obtained all the variants needed to carry out next calculation by using (12) and (13).

With the distributed working style, we present a theorem, which essentially highlights some important properties of our scheduling strategy.

Theorem 2 (*Cycle free*). *An optimal solution to problem* (5) *is cycle free, i.e., there exists no load transfer such that* $x_{i_1 i_2} > 0, x_{i_2 i_3} > 0, \ldots, x_{i_m i_1} > 0$, *where* $i_1, \ldots, i_m \in N$.

Proof. We assume that in rth iteration, i_1 has some loads to send to its neighboring nodes. Node i uses (12) and (13) to search an optimal solution for its sub-graph. If $x_{i_1 i_2}^{(r)} > 0$, the path $(i_1 \to i_2 \to d)$ must be the MDRP in P_{i_0} and $d_{i_1}^{0(r-1)} > d_{i_2}^{0(r-1)}$ holds that is, $T_{i_1}^{(r-1)} > T_{i_2}^{(r-1)}$. Similarly, if $x_{i_2 i_3}^{(r)} > 0$, we can also obtain $d_{i_2}^{0(r-1)} > d_{i_3}^{0(r-1)}$. Recursively, we observe that the following inequalities hold:

$$d_{i_1}^{0(r-1)} > d_{i_2}^{0(r-1)} > \cdots > d_{i_m}^{0(r-1)}.$$

For node i_m, if it should send some loads to node i_1, the inequality $d_{i_m}^{0(r-1)} > d_{i_1}^{0(r-1)}$ must hold, which is a clear contradiction. Hence, in each iteration, it is impossible that a load cycle $x_{i_1 i_2}^{(r)} > 0, x_{i_2 i_3}^{(r)} > 0, \ldots, x_{i_m i_1}^{(r)} > 0$ exists. Our optimal solution is approached by the iterations and it can be deduced that Theorem 2 holds. □

3.2.2. Discussions on divisible loads originating from multiple sites

Though each node i can calculate the local optimal solution according to $\gamma_i^{(r)}$ in rth iteration, where $\gamma_i^{(r)} = L_i + \sum_{j \in V_i} x_{ji}^{(r)}$ and node i also can handle the situation when $L_i > 0$, up to now we only assume that one of the nodes in the system has some loads to be scheduled initially. Because of the assumption that each node can start processing loads only after it receives all loads coming from its neighboring nodes, we may meet an interesting problem when we use the search method for the divisible loads originating from two or more sites.

In some cases, some node may not receive or send loads to its neighboring nodes at all and the optimal solution will never be achieved. For example, we only consider two nodes in a system, i and j, which are connected by links (i, j) and (j, i), and $L_i > 0, L_j > 0$. Node i may have other neighboring nodes and node j has only one neighboring node. Initially we assume $E_i \times L_i > E_j \times L_j$, and for the optimal solution, some loads shall be transferred from node i to j. However, node j cannot start processing load until it receives all loads from node i. For node i, it has many routing paths including $(i \to d)$ and $(i \to j \to d)$ and the time delays of the two paths should be $d_i^0 = E_i \times l_i$ and $d_i^m = t_j^{\max} + E_j \times (L_j + x_{ij})$, where $t_j^{\max} = \sum_{k=1}^{m} (C_{in} \times x_{in}), n = S_i^{(-1)}(k), m = S_i(j)$. Due to $t_j^{\max}$, the time instance at which the last job arrives at node j and L_j, d_i^m may be greater than d_i^0. Hence node i will not send loads to node j. For node j, needless to mention that it will not send loads to node i, because of $d_j^0 < d_j^1$, where path 1 is $(j \to i \to d)$.

We refer to this problem as a "Trap" problem. In order to solve "Trap" problem, we shall relax our assumption that each node cannot start processing until it receives all loads from its neighboring nodes. For node i, we assume that node i can start processing L_i (locally processing) at $t = 0$. When the first load arrives at node i, it will stop processing and start to receive loads. Node i can resume the processing after the last job arrives at node i. According to this modification, we must determine the time instance at which the first load arrives at each node. After the last job is received by node i, we obtain the following:

$$t_i^{\min} = \min \left\{ t_j^{\max} + \sum_{k=1}^{\bar{S}_j(i)-1} C_{j\bar{S}_j^{-1}(k)} \times x_{j\bar{S}_j^{-1}(k)}, \; |\forall j \in V_i, x_{ji} > 0 \right\}, \tag{19}$$

$$t_{i-j}^{\min} = \min \left\{ t_m^{\max} + \sum_{k=1}^{\bar{S}_m(i)-1} C_{mn} \times x_{mn}, |\forall m \in V_i, m \neq j, x_{mi} > 0, n = \bar{S}_m^{-1}(k) \right\}.$$

When node i resumes processing, it has some unprocessed load, which is given by

$$L_i^* = \max \left\{ 0, L_i - \frac{t_i^{\min}}{E_i} \right\}.$$

Then, in each iteration, the total amount of loads $\gamma_i^{(r)}$ should be changed to

$$\gamma_i^{(r)} = L_i^{*(r)} + \sum_{j \in V_i} x_{ji}^{(r)}, \; L_i^{*(r)} = \max \left\{ 0, L_i - \frac{t_i^{\min(r)}}{E_i} \right\}. \tag{20}$$

After this modification, each node i can calculate the time delay of all the routing paths according to $L^{*(r)}$, thus solving the "Trap" problem.

The distributed working style of the proposed scheduling strategy is as follows. At initial time, a processor starts to compute an optimal solution for its sub-graph and also informs the rest of the nodes to compute one by one in a pre-defined sequence (order), which can be any potential sequence in the topology. This processor is referred to as a *core* processor. The objective of the core processor is to avoid the time asynchrony among the processors in the system and to determine the time instances when to trigger computations and when to stop computing. When node i finishes computing and obtains a solution it broadcasts the solution to its neighboring nodes. After some iterations, when the stopping rule is satisfied, all the

Table 1
Proposed scheduling strategy for loads originating from single and multiple sites

Step 1: ***Initialization***

$r = 0$ (r: iteration index).

For $i = 1$ to n, $n = |N|$.

1.1 Determine P_i, according to the optimal sequency S_i.

1.2 Calculate the total amount loads at node i, which is $\gamma_i^{(0)} = L_i + \sum_{j \in V_i} x_{ji}^{(0)}$.

1.3 Find a feasible set $x_i^{(0)}$ as an initial feasible solution, which satisfies $\sum_{k=0}^{v_i} x_i^{k(0)} = \gamma_i^{(0)}$. Here, we choose $x_i^{(0)} = \gamma_i^{(0)}$, $x_i^{k(0)} = 0, k = 1, \ldots, v_i$.

End For.

Step 2: ***Solution Procedure***

Let $r = r + 1$.

For $i = 1$ to $n, n = |N|$, do the following for node i.

2.1 If $L_i > 0$, calculate $t_i^{\min(r-1)}$ by (19) and $\gamma_i^{(r)}$ by (20).

Else, $\gamma_i^{(r)} = \sum_{j \in V_i} x_{ji}^{(r-1)}$.

2.2 Determine $t_i^{\max(r)}$ according to the incoming loads $x_{ji}^{(r-1)}$, and $d_i^{0(r-1)}$.

2.3 For $k = 1$ to v_i, do the following for each neighboring node $j = S_i^{-1}(k)$.

If $x_{ji}^{(r-1)} > 0$, do the next loop.

Calculate $t_j^{\max(r-1)}$, $t_{j-i}^{\max(r-1)}$, and $t_j^{\min(r-1)}$, $t_{j-i}^{\min(r-1)}$ if $L_j > 0$.

Calculate d_i^k and $\frac{\partial d_i^k}{\partial x_{ij}^{(r-1)}}$, by (17) and (18), respectively.

Find MDRP $d_i^{\min} = \min\{d_i^k\}$.

End For.

2.4 For $k = 0$ to v_i, do the following for each routing path.

Let $x_i^{k(r)} = \max\{0, x_i^{k(r-1)} - \alpha^{(r)}(d_i^k - d_i^{\min})/H_i^k\}, k \neq \min$, and

$x_i^{\min(r)} = \gamma_i^{(r)} - \sum_{k=0, k \neq \min}^{v_i}(x_i^{k(r)})$.

End For.

End For.

Step 3: ***Stopping Rule***

If $|T^{\max(r)} - T^{\min(r)})| < \varepsilon$, then **stop**, where, $T^{\max(r)} = \max\{T_i^{(r)}, |\forall i \in N\}$, $T^{\min(r)} = \min\{T_i^{(r)}, |\forall i \in N\}$, and ε is a desired acceptable tolerance for solution quality; otherwise, go to Step 2.

nodes stop computing and obtain an optimal solution of the system. Our proposed algorithm can be also implemented in a centralized method. If a processor has all the information about the topology of the system and about the processors and links, it can simulate the working style of each node and search for an optimal solution of the system. Once the node obtains the optimal solution, it can inform the rest of nodes to transfer the loads according to the optimal scheduling. Thus, our scheduling strategy is flexible for both distributed situation, when a node in the system has knowledge only about its local topology, and centralized situation, when one key node has the information about the whole system.

We have proposed our scheduling strategy for divisible loads originating from single and multiple sites. A pseudo-code of our proposed algorithm is also shown in Table 1. Note that, for each node searching for a local solution, there is little difference between the sub-algorithm proposed in Section 3.1 and the search method used in our strategy. In [4,3], it is pointed out that to obtain a faster convergence rate, it is better to reach a solution between the original starting point and the local optimal point in each iteration instead of reaching the local optimal point directly. Hence, in our strategy, in each system iteration r, every node calculates the amount of loads on every feasible routing path only once, instead of several times as shown in Section 3.1.

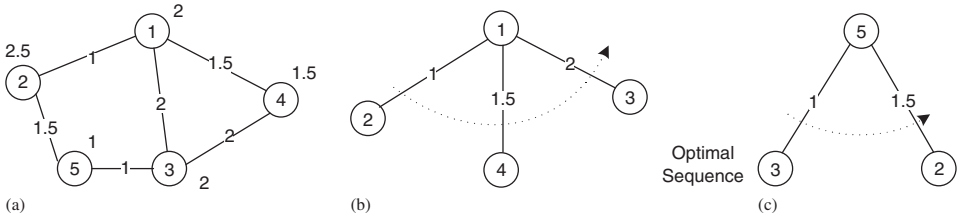

Fig. 2. Experiment 1: (a) an arbitrary network; (b) optimal sequence of node 1; (c) optimal sequence of node 5.

3.3. Complexities of the algorithm

We shall now quantify the complexities of the algorithm proposed in this paper. As a first step of our algorithm, we sort the neighboring nodes of node i in the system according to optimal sequence. We use Quicksort algorithm [13] which takes us $O(v_i \lg v_i)$ steps for node i, where v_i is the number of node i's neighboring nodes ($v_i = |V_i|$). Thus, it consumes $O(\max\{v_i \lg v_i, |\forall i \in N\} \times n)$ steps for the system to obtain the optimal sequence for each node. Considering $v_i \leqslant (n-1), n = |N|$, we can simplify this complexity to $O(n^2 \lg n)$. Note that, once the optimal sequence is determined, all the calculations can follow this sequence. Further, in each iteration r, in our calculations, we shall compute the amount of loads on each possible routing path of node i which takes us $O(v_i + 1)$ steps. The complexities to calculate in one iteration should be $O((v_1 + 1) + \cdots + (v_n + 1)) \approx O(n^2)$. Because our algorithm can converge to an optimal solution after some iterations R (please refer to Appendix A for more details about R), the complexity of the computation can be $O(Rn^2)$ steps. Hence, the total complexity of our algorithm is $O(n^2 \lg n) + O(Rn^2) \approx O(n^2 \lg n)$.

4. Simulation results and some discussions

Our proposed algorithm retains three important features of practical interest: firstly, the algorithm is simple to realize and can be implemented in a distributed fashion. The second one is in its style of working by avoiding the need for generating a timing diagram explicitly for any complex networks having an arbitrary network topology. The last one is its capability of handling divisible loads originating from both single and multiple sites. To the best of our knowledge, the studies attempted in [27] are closest to our work. In [27], the authors proposed a centralized algorithm named "*Resource-Aware Optimal Load Distribution Algorithm*" (RAOLD) to solve the scheduling problem of divisible load originating from a single site on arbitrary graphs. It will be interesting to compare the performance of RAOLD and our algorithm and discuss the main difference of the two algorithms in the case that divisible loads originating from a single site on arbitrary networks. Further, we will consider divisible loads originating from multiple sites on several types of networks such as *sparse*, *medium-dense* and *dense* so as to capture the performance of our algorithm.

4.1. Demonstration of proposed algorithm

Following example clarifies and illustrates the workings of our proposed algorithm in seeking optimal solution, if it exists.

Experiment 1. Consider a distributed computer system that consists of five host computers (nodes), as illustrated in Fig. 2(a). There are $L = 100$ units divisible loads originating from node 1. The processor speed parameters and the respective link speed parameters are randomly generated and are marked in the graph. In this experiment, node 1 will try to send some loads to rest of the nodes to obtain a minimum processing time.

As a first step of our proposed algorithm, we determine an optimal sequence $\bar{S}_i$ defined in Theorem 1 for node i, $i = 1, 2, \ldots, 5$. For example, the optimal sequence of node 1 and node 5 are shown in Fig. 2(b) and (c), respectively. Then, we determine the routing path of each node i according to the optimal sequence $\bar{S}_i$ one by one with the virtual node d and the virtual links (i, d) and (j, d), $j \in V_i$. Each node i obtains a set of routing paths $P_i = [p_i^0, p_i^1, \ldots, p_i^{v_i}]$, $v_i = |V_i|$. For example, node 2 has three routing paths, which are $p_2^0 = (2 \to d)$, $p_2^1 = (2 \to 1 \to d)$ and $p_2^2 = (2 \to 5 \to d)$. From system setting, we can obtain that $\gamma_1^{(0)} = L = 100$ and $\gamma_2^{(0)} = \gamma_3^{(0)} = \gamma_4^{(0)} = \gamma_5^{(0)} = 0$. After this, we set a feasible solution in the first iteration, which is $x_1^{(0)} = [x_1^{0(0)}, \ldots, x_1^{3(0)}] = [100, 0, 0, 0]$, $x_2^{(0)} = [x_2^{0(0)}, \ldots, x_2^{2(0)}] = [0, 0, 0]$, and so on. This completes the *Initialization* step in our proposed algorithm. Then, in the *Solution Procedure* (second) step, each node i will calculate the amount of loads on each path in P_i according to the equations discussed in Section 3.1 and 3.2. Note that in each iteration r, node i will skip the calculation on path $(i \to j \to d)$, if $x_{ji}^{(r)} > 0$, $j \in V_i$. When the rth iteration is completed, the algorithm will determine $T^{\max(r)}$ and $T^{\min(r)}$ (the maximum processing time and the minimum processing time among all the processors, respectively). If the difference between $T^{\max(r)}$ and $T^{\min(r)}$ is within an acceptable tolerance ε, we stop the *Solution Procedure* and obtain a near-optimal solution. [1]

[1] Note that the optimal solution is obtained when all nodes stop computing at the same time instant, as shown in the literature [6] and in (11) above. However, in our iterative procedure, it is natural that we strike a near-optimal solution due to the stopping rule with $\varepsilon > 0$.

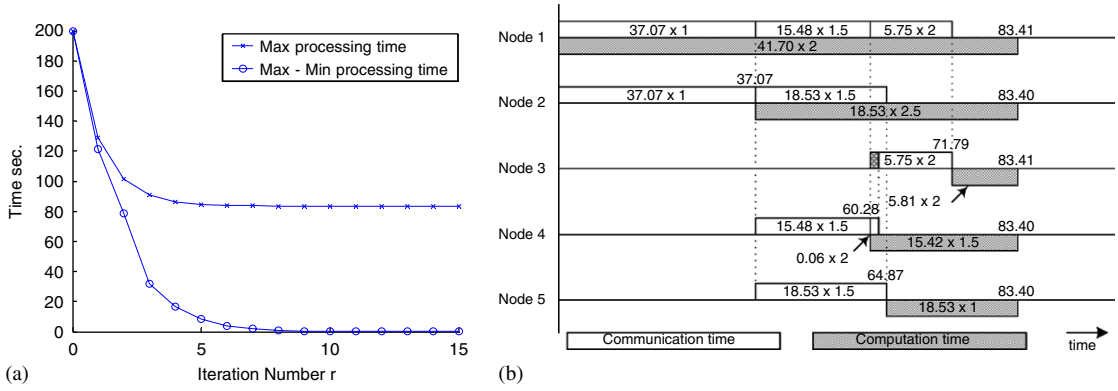

Fig. 3. Results of experiment 1: (a) $T^{\max(r)}$ and $T^{\max(r)} - T^{\min(r)}$ in each iteration r; (b) timing diagram of the system.

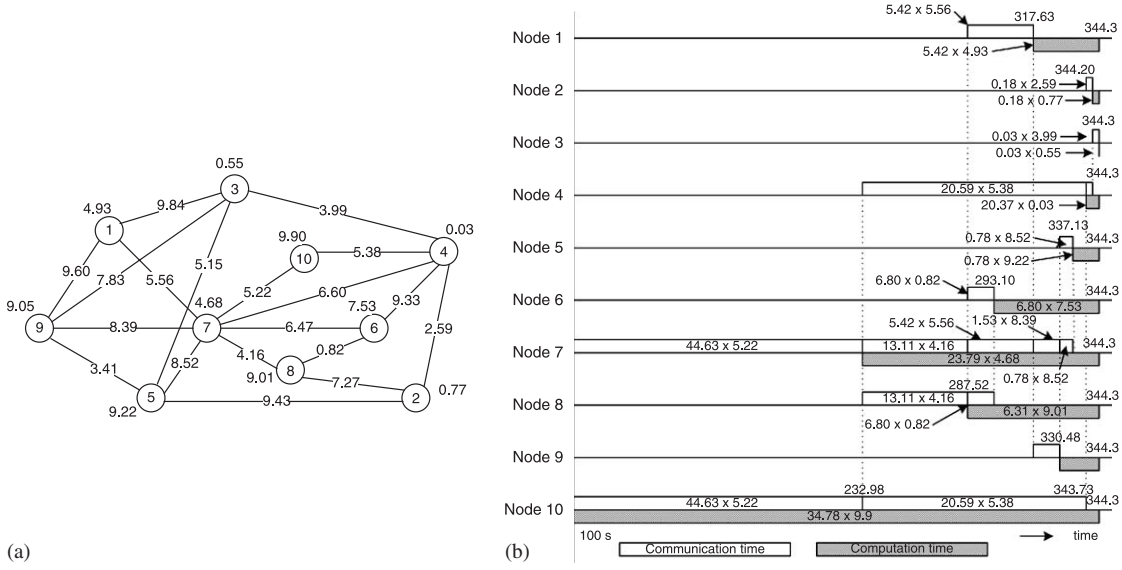

Fig. 4. Experiment 2: (a) an arbitrary network with 10 nodes; (b) timing diagram of the proposed algorithm when divisible loads originate from node 10.

Otherwise, continue the procedure. In all experiments in this paper, we set $\varepsilon = 0.01$ s.

In Fig. 3(a), we show the maximum processing time and the difference between the maximum and minimum processing time of each iteration r for Experiment 1. From this figure, we can observe that in this experiment the maximum processing time converges to 83.41 s, and the difference between the maximum and minimum processing time converges to 0. After 15 iterations, we obtain an optimal solution for this system, which is $\bar{l}_1 = 41.70, \bar{l}_2 = 18.53, \bar{l}_3 = 5.81, \bar{l}_4 = 15.42, \bar{l}_5 = 18.53$. The timing diagram of the solution is also shown in Fig. 3(b). By using the proposed algorithm, it is shown that the optimal processing time of the system is 83.41 s, which is much shorter than the processing time 200 s in the case of no loads scheduling, where only node 1 processes its own loads.

Table 2
Computational results for load distribution with different load origination

Root node (i)	RAOLD (s)	Proposed algorithm (s)	Gain (%)
1	3.048×10^2	2.797×10^2	8.2
2	5.951×10^1	5.951×10^1	0.0
3	4.454×10^1	4.451×10^1	0.1
4	2.969×10^0	2.968×10^0	0.0
5	3.139×10^2	2.966×10^2	5.5
6	1.671×10^2	1.514×10^2	9.4
7	2.340×10^2	2.319×10^2	0.9
8	1.234×10^2	1.214×10^2	1.6
9	3.913×10^2	3.253×10^2	16.9
10	4.300×10^2	3.443×10^2	19.9

4.2. Performance comparison of algorithms

Notice that if divisible loads are originating from a single site, the RAOLD algorithm proposed in [27] can also be applied directly to solve the scheduling problem. We shall first give a brief introduction of RAOLD algorithm and then compare the proposed algorithm with RAOLD algorithm in terms of processing time of the system.

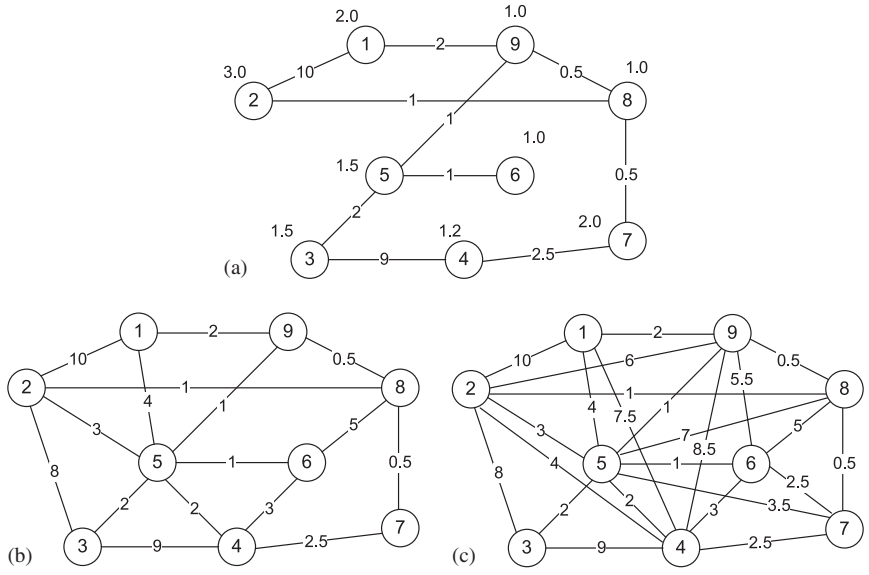

Fig. 5. Experiment 3, a nine nodes system: (a) a sparse connection; (b) a medium-dense connection; (c) a dense connection.

Algorithm **RAOLD** [27]: In this work, the authors transformed the scheduling problem into a multi-level unbalanced tree network. For a given arbitrary network, RAOLD algorithm first generates a *minimum cost spanning tree* (MST) by using Kruskal's or Prim's algorithms [4,13]. The root node of the MST is the node at which a divisible load is assumed to originate. Then, RAOLD algorithm applies **Rule A** in [6] systematically in a *bottom-up* fashion to the MST, starting from the last level to the root, to obtain an optimal reduced tree, which is a single-level tree. Following the optimal sequence defined in Theorem 1, RAOLD algorithm can deliver an optimal time performance. Interested readers can refer to [27] for more details. Now we conduct the following experiment to compare the performance of the proposed algorithm with RAOLD algorithm.

Experiment 2. Consider a distributed computer system that consists of 10 host processors (nodes), as illustrated in Fig. 4(a). The processor speed and the link speed parameters are chosen using a uniform probability distribution in the range [0.01, 10] and they are marked in this graph. We assume $L = 100$. Now, as per our proposed algorithm, we construct 10 sub-experiments, in which the divisible loads are assumed to originate at node 1, 2, . . ., 10, respectively. The experimental results are shown in Table 2.

From Table 2, we can observe that in the cases tested, our proposed algorithm is at least as good as RAOLD algorithm and sometimes up to 20% better in terms of the processing time of the system. In Fig. 4(b), we present the timing diagram of the proposed algorithm when the divisible loads originate from node 10.

It would be of natural interest to investigate the mechanics behind the gain achieved by our algorithm over RAOLD. The reason is as follows. RAOLD algorithm is based on the MST generated from the original arbitrary topology. MST is only determined by the cost of links in the network. However, the number of hops that the loads shall be retransferred also significantly influences the processing time of the system. In some cases, if we choose the minimum number of hop instead of minimum cost of links, we can improve the system performance significantly. Our algorithm takes advantage of both the cost of links and the number of hops, and hence performs much better than RAOLD which only considers the cost of links.

Table 3
Computational results of experiment 3 (unit load)

Node i	Sparse network	Medium dense network	Dense network
1	6.42	4.84	5.30
2	7.69	5.27	1.18
3	33.34	32.09	32.07
4	3.65	5.08	8.55
5	17.12	12.92	14.2
6	50.01	48.13	48.11
7	0.97	13.66	12.9
8	30.78	29.87	29.56
9	50.01	48.14	48.11
Processing time	50.01 s	48.14 s	48.11 s

4.3. Divisible loads originating from multiple sites

In this section, we shall conduct some experiments on divisible loads originating from multiple sites on arbitrary networks. Without loss of generality, we set the number of nodes in the system to nine. By varying the link probability as 25%, 50% and 80%, we generate sparse, medium-dense, and dense networks, respectively. That is, the connectivity is varied from 25% to 80% to capture different types of graphs as shown in Fig. 5.

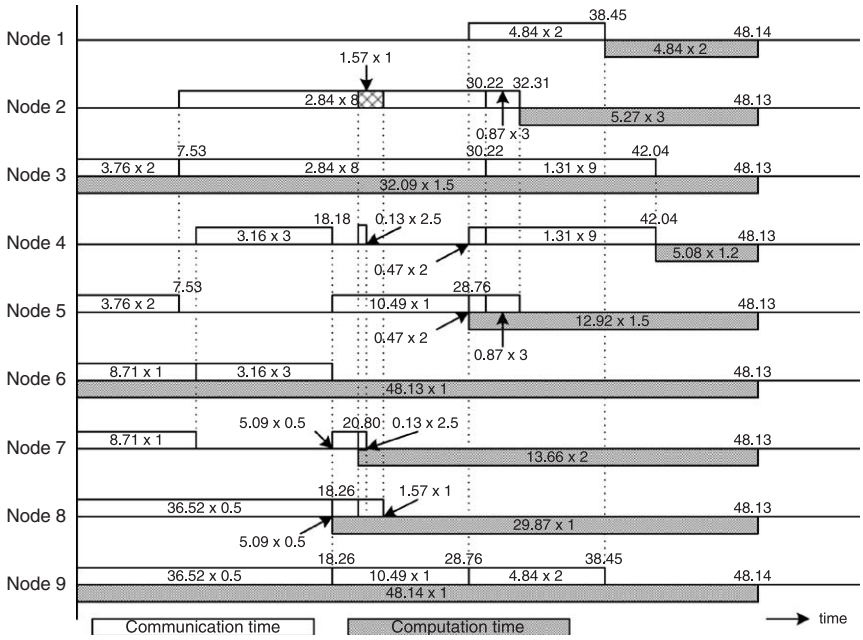

Fig. 6. The timing diagram of the 9-node system with medium dense connectivity.

Experiment 3. Consider an arbitrary network system with nine nodes. We assume that the divisible loads originate at node 3, 6 and 9 and we set $L_3 = 40$, $L_6 = 60$ and $L_9 = 100$ unit load throughout this experiment. We randomly generate a sparse network, a medium-dense network and a dense network, as shown in Fig. 5(a)–(c), respectively. The process speed parameters of the nodes are marked in 5(a) and keep the same throughout this experiment. In each network, the link speed parameters are randomly chosen according to a uniform distribution in the range [0.5, 10] and the values are also marked in the graphs.

We conduct three sub-experiments to test the performance of our proposed algorithm for these networks and we show the results in Table 3, which presents the optimal solution $\bar{l}_i$ in unit load for each node i. From Table 3, we observe that the system processing time decreases by increasing the degree of connectivity. In Fig. 6, we show that the timing diagram of the system when network connectivity is medium dense. Through these experiments, we can draw a conclusion that our proposed algorithm can also obtain the optimal solutions efficiently in the cases when divisible loads originate from multiple sites.

5. Conclusions

In this paper, we have studied an optimal scheduling problem of divisible loads originating from single or multiple sites in a distributed computer system with arbitrary network configurations. We have presented a generic mathematical model for this problem and formulated it as a real-valued constrained optimization problem. We decomposed the problem into sub-problems which could be solved by each processor in the system. We have derived the *necessary and sufficient* conditions for the optimal solution of the sub-problems and we have proven that after some iterations, our solution can converge to a global optimal solution of the system. On the basis of this study, we have proposed a *distributed* algorithm to solve the general divisible loads scheduling problem. We have adopted a load balancing approach using the concept of virtual routing in the design of our distributed algorithm. We analyzed the difference between the cases that divisible loads originate from single and multiple sites in our proposed algorithm and proposed a simple model to solve the "Trap" problem that occurs in the case of multiple sites. From the extensive simulations, we have proven that our proposed algorithm can efficiently solve the scheduling problem of divisible loads originating from single and multiple sites on arbitrary networks.

The approach followed in this paper is a novel contribution to the domain of DLT. We designed a distributed scheduling strategy to achieve the optimal processing time of all loads in the system. It is the first time that a distributed algorithm is attempted and proposed in the DLT literature. The main advantage of this approach comes from the fact that the need for generating a timing diagram to develop recursive equations is altogether circumvented. In this paper, we have taken a radically different approach to address this complex problem by carefully formulating as a generalized minimization problem, thus avoiding the need for a timing diagram representation.

Here, we would like to mention some plausible extensions to this work. One of the immediate extensions could be in studying the performance of the proposed algorithm on large-scale networks, such as grid computing system, as availability of the resources are non-deterministic. Further, in this paper, the delay functions are assumed to be linear. It would be interesting to consider non-linear functions for the delay components in the problem formulation. This seems to be possible as our optimization formulation presented in this paper is generic.

Appendix A. The estimation of R

To estimate the complexity of R, we consider a system with n nodes. Obviously, the worst case is the linear daisy chain network with the load originating from node 1, as shown in Fig. 7(a). Before we start our algorithm, we can observe that $T_1^{(0)} = L_1 \cdot E_1$ and $T_2^{(0)} = T_3^{(0)} = \cdots = T_n^{(0)} = 0$, as shown in Fig. 7(b). In each iteration of our algorithm, we let node 1 start computation first. When node 1 finishes, node 2 starts, and so on till node n finishes its computation. Now, we denote $\triangle_i^{(r)} = T_{i-1}^{(r)} - T_i^{(r)}$ as the time difference between node $i-1$ and node i in iteration r, where $1 < i \leqslant n$ and $\triangle_i^{(r)} \geqslant 0$ always holds. When $r = 0$, we can obtain $\triangle_2^{(0)} = E_1 \cdot L_1$ and $\triangle_i^{(0)} = 0, 2 < i \leqslant n$. After first iteration, we can obtain the following equations:

$$\begin{aligned}
\triangle_2^{(1)} &= \frac{E_1}{C_{12}+E_1+E_2} \cdot \frac{E_2}{C_{23}+E_2+E_3} \cdot E_2 \cdot L_1 \\
&= \frac{E_2}{C_{12}+E_1+E_2} \cdot \frac{E_2}{C_{23}+E_2+E_3} \cdot \triangle_2^{(0)}, \\
\triangle_3^{(1)} &= \frac{E_1}{C_{12}+E_1+E_2} \cdot \frac{E_2}{C_{23}+E_2+E_3} \\
&\quad \cdot \frac{E_3}{C_{34}+E_3+E_4} \cdot E_3 \cdot L_1 \\
&= \frac{E_3}{C_{34}+E_3+E_4} \cdot \frac{E_3}{E_2} \cdot \triangle_2^{(1)}, \\
&\vdots \\
\triangle_{n-1}^{(1)} &= \prod_{i=1}^{n-1} \frac{E_i}{C_{i(i+1)}+E_i+E_{i+1}} \cdot E_{n-1} \cdot L_1 \\
&= \prod_{i=3}^{n-1} \frac{E_i}{C_{i(i+1)}+E_i+E_{i+1}} \cdot \frac{E_{n-1}}{E_2} \cdot \triangle_2^{(1)}, \\
\triangle_n^{(1)} &= 0.
\end{aligned}$$

We define $\triangle^r = T_1^{(r)} - T_n^{(r)}$ as the $T^{\max(r)} - T^{\min(r)}$, which can be deduced as

$$\begin{aligned}
\triangle^{(r)} &= \sum_{i=2}^{n-1} \triangle_i^{(r)} \\
&= \left[1 + \sum_{i=3}^{n-1} \left(\prod_{j=3}^{i} \frac{E_j}{C_{j(j+1)}+E_j+E_{j+1}} \cdot \frac{E_i}{E_2} \right) \right] \cdot \triangle_2^{(r)} \\
&= \phi \cdot \triangle_2^{(r)}, \qquad (21)
\end{aligned}$$

where ϕ is a fixed number according to the network settings. In iteration r, we can deduce that $\triangle_2^{(r)}$ is

$$\begin{aligned}
\triangle_2^{(r)} &= \left(\frac{E_2}{C_{12}+E_1+E_2} \cdot \triangle_2^{(r-1)} + \triangle_3^{(r-1)} \right) \cdot \frac{E_2}{C_{23}+E_2+E_3} \\
&= \left(\frac{E_2}{C_{12}+E_1+E_2} \cdot \frac{E_2}{C_{23}+E_2+E_3} + \frac{E_3}{C_{23}+E_2+E_3} \right. \\
&\quad \left. \cdot \frac{E_3}{C_{34}+E_3+E_4} \right) \cdot \triangle_2^{(r-1)} \\
&= \eta \cdot \triangle_2^{(r-1)}. \qquad (22)
\end{aligned}$$

Obviously, $0 < \eta < 1$ holds. By (21) and (22), we obtain the following equation:

$$\begin{aligned}
\triangle^{(r)} &= \phi \cdot \triangle_2^{(r)} \\
&< \lim_{n\to\infty} \left[1 + \sum_{i=3}^{n-1} \left(\prod_{j=3}^{i} \frac{E_j}{C_{j(j+1)}+E_j+E_{j+1}} \cdot \frac{E_i}{E_2} \right) \right] \\
&\quad \cdot \triangle_2^{(r)} = \Phi \cdot \triangle_2^{(r)} \\
&\Rightarrow \triangle^{(r)} < \Phi \cdot \triangle_2^{(r)} \leqslant \Phi \cdot \eta^{r-1} \cdot E_1 \cdot L_1. \qquad (23)
\end{aligned}$$

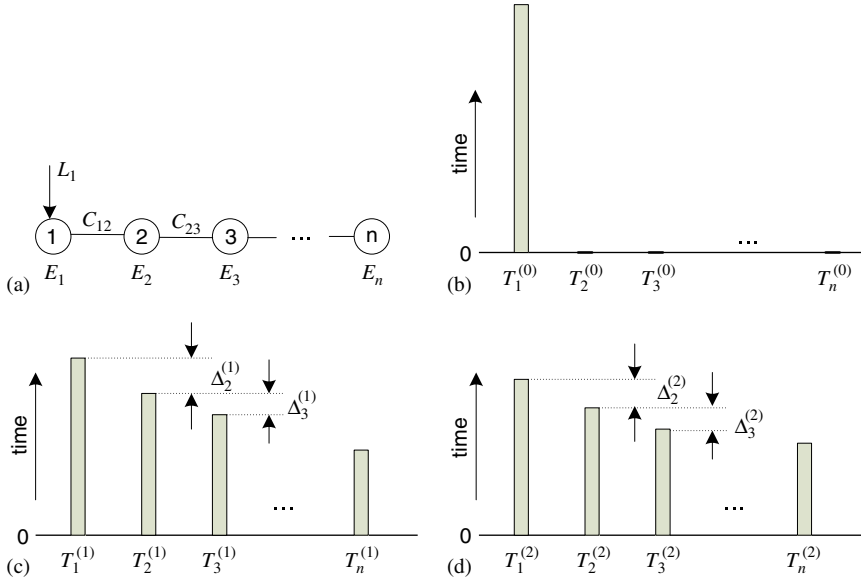

Fig. 7. The iteration procedure of the proposed algorithm.

From (23), we can see R is independent of n. Hence, if we determine the value of ε, we can estimate the iteration number R by

$$R > \frac{\ln \varepsilon - \ln \Phi - \ln E_1 - \ln L_1}{\ln \eta} + 1. \quad (24)$$

Now, we should consider the case when divisible loads originate from multiple sites. In our model, we assume that the distributed system is a linear system and each node can determine its scheduling strategy independently. We can consider the system as the combination of several sub-systems and in each sub-system, there is only one node with non-zero divisible loads initially. As we proved before, each sub-system can converge to a global optimal solution in R iterations. In each iteration, nodes will calculate the scheduling strategy for every sub-system, obtain the solutions for all of the sub-systems and then add them together. Hence, we can draw a conclusion that when divisible loads originate from multiple sites, the iteration number R is no more than that when divisible loads originate from a single site.

Appendix B. The list of notations

N	the set of nodes in the system
n	the total number of nodes in the system
W	the set of links in the system
V_i	the set of neighboring nodes of node i, where $i = 1, 2, \ldots, n$
v_i	the number of neighboring nodes of node i, $v_i = \|V_i\|$
L_i	the amount of loads originating at node i for processing, where $L_i \geqslant 0$
l_i	the amount of loads assigned to node i to process
x_{ij}	the amount of loads transferred from node i to node j, $j \in V_i$
γ_i	the total amount of loads at node i
E_i	the time taken to compute a unit load by node i
C_{ij}	the time taken to transfer a unit load on link (i, j)
$t_i^{\max}$	the time instance at which the last load is received by node i
$t_i^{\max}(j)$	the time instance at which the last load coming from node j is received by node i
$t_{i-j}^{\max}$	the time instance at which the last load coming from V_i, except node j, is received by node i
$t_i^{\min}$	the time instance at which the first load arrives at node i
$t_{i-j}^{\min}$	the time instance at which the first load coming from V_i, except node j, is received by node i
S_i	the transfer sequence following which node i sends loads to its neighboring nodes
S_i^{-1}	the inverse function of S_i for node i
P_i	the set of routing paths of node i
p_i^k	the kth routing path in P_i
d_i^k	the time delay on path p_i^k

References

[1] R. Agarwal, H.V. Jagadish, Partitioning techniques for large-grained parallelism, IEEE Trans. Comput. 37 (12) (1998) 1627–1634.

[2] 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 Distributed Systems 9 (5) (1998) 429–441.

[3] D.P. Bertsekas, Nonlinear Programming, Athena Scientific, Belmont, MA, 1995.

[4] D.P. Bertsekas, R. Gallager, Data Networks, Prentice-Hall Inc., 1992.

[5] V. Bharadwaj, G. Barlas, Efficient scheduling strategies for processing multiple divisible loads on bus networks, J. Parallel Distributed Comput. 62 (1) (2002) 132–151.

[6] V. Bharadwaj, D. Ghose, V. Mani, T.G. Robertazzi, Scheduling Divisible Loads in Parallel and Distributed Systems, IEEE Computer Society Press, Los Almitos, CA, 1996.

[7] V. Bharadwaj, D. Ghose, T.G. Robertazzi, Divisible load theory: a new paradigm for load scheduling in distributed system, Special Issue on Divisible Load Scheduling in Cluster Computing, vol. 6, no. 1, Kluwer Academic Publishers, Dordrecht, 2003, pp. 7–18.

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

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

[10] J. Blazewicz, M. Drozdowski, M. Markiewicz, Divisible task scheduling-concept and verification, Parallel Comput. 25 (1999) 87–98.

[11] S.K. Chan, V. Bharadwaj, D. Ghose, Large matrix–vector products on distributed bus networks with communication delays using the divisible load paradigm: performance analysis and simulation, Math. Comput. Simulation 58 (2001) 71–79.

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

[13] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms, The MIT Press, Cambridge, MA, 1994.

[14] S.C. Datermos, F.T. Sparrow, The traffic assignment problem for a general network, Transportation Sci. 6 (1969) 73–87.

[15] M. Drozdowski, Selected Problems of Scheduling Tasks in Multiprocessor Computer Systems, Wydawnictwa Politechniki Poznanskiej (In English) Book no. 321, Poznan, Poland, 1997.

[16] M.M. Eshaghian, Heterogeneous Computing, Artech House, 1996.

[17] D. Ghose, H.J. Kim, H.T. Kim, Adaptive divisible load scheduling strategies for workstation clusters with unknown network resources, Technical Report KNU/CI/MSL/001/2003, 2003.

[18] W. Glazek, A multistage load distribution strategy for three-dimensional meshes, Special Issue on Divisible Load Scheduling in Cluster Computing, vol. 6, no. 1, Kluwer Academic Publishers, Dordrecht, January 2003, pp. 31–40.

[19] J.T. Hung, H.J. Kim, T.G. Robertazzi, Scalable scheduling in parallel processors, Conference on Information Sciences and Systems, Princeton University, March 20–22, 2002.

[20] K. Li, Parallel processing of divisible loads on partitionable static interconnection networks, Special Issue on Divisible Load Scheduling in Cluster Computing, vol. 6, no. 1, Kluwer Academic Publishers, Dordrecht, January 2003, pp. 47–56.

[21] X. Li, V. Bharadwaj, C.C. Ko, Scheduling divisible tasks on heterogeneous single-level tree networks with finite-size buffers, IEEE Trans. Aerospace Electron. Systems 37 (2001) 1298–1308.

[22] J. Li, H. Kameda, Load balancing problems for multiclass jobs in distributed/parallel computer systems, IEEE Trans. Comput. 47 (3) (1998) 322–332.
[23] T.G. Robertazzi, Ten reasons to use divisible load theory, Computer 36 (5) (2003) 63–68.
[24] J. Sohn, T.G. Robertazzi, Optimal divisible job load sharing on bus networks, IEEE Trans. Aerospace Electron. Systems 1 (1996).
[25] D.J. Wilde, C.S. Beightler, Foundations of Optimization, Prentice-Hall, Englewood Cliffs NJ, 1967.
[26] H.M. Wong, D. Yu, V. Bharadwaj, T.G. Robertazzi, Data intensive grid scheduling: multiple sources with capacity constraints, In the Proceedings of the International Conference on International Conference on Parallel and Distributed Computing and Systems, Marina del Rey, CA, November 2003.
[27] J. Yao, V. Bharadwaj, Design and performance analysis of divisible load scheduling strategies on arbitrary graphs, Cluster Computing 7 (2) (2004) 191–207.
[28] Z. Zeng, V. Bharadwaj, Design and analysis of a non-preemptive decentralized load balancing algorithm for multi-class jobs in distributed networks, Comput. Commun. 27 (2004) 679–694.

Zeng Zeng received the Ph.D. in electrical and computer engineering from The National University of Singapore, in 2005. He received his BS and MS degrees in automatic control from Huazhong University of Science and Technology, Wuhan, PR China, in 1997 and 2000, respectively. His research interests include distributed/grid computing systems, multimedia storage systems, wireless sensor networks, and controller area networks. Currently, he works as a research fellow at The National University of Singapore.

Bharadwaj Veeravalli, Member, IEEE and IEEE-CS, received his BSc in Physics, from Madurai-Kamaraj University, India in 1987, Master's in Electrical Communication Engineering from Indian Institute of Science, Bangalore, India in 1991 and PhD from Department of Aerospace Engineering, Indian Institute of Science, Bangalore, India in 1994. He did his post-doctoral research in the Department of Computer Science, Concordia University, Montreal, Canada, in 1996. He is currently with the Department of Electrical and Computer Engineering, Communications and Information Engineering (CIE) division, at The National University of Singapore, as a tenured Associate Professor. His main stream research interests include, Multiprocessor systems, Cluster/Grid computing, Scheduling in parallel and distributed systems, Bioinformatics and Computational Biology, and Multimedia computing. He is one of the earliest researchers in the field of divisible load theory (DLT). He has published over 85 papers in high-quality International Journals and Conferences. He had successfully secured several externally funded projects. He is currently serving the Editorial Board of IEEE Transactions on Computers, IEEE Transactions on SMC-A, and International Journal of Computers & Applications, USA, as an Associate Editor.