Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/090539

Priority-based Divisible Load Scheduling using Analytical Hierarchy Process

Shamsollah Ghanbari[1,*], *Mohamed Othman*[1,2,*], *Mohd Rizam Abu Bakar*[1] *and Wah June Leong* [1]

[1] Laboratory of Computational Science and Mathematical Physics, Institute For Mathematical Research, Universiti Putra Malaysia 43400 UPM Serdang, Selangor D.E., Malaysia
[2] Dept of Communication Tech and Network, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia 43400 UPM Serdang, Selangor D.E., Malaysia

Received: 7 Feb. 2015, Revised: 9 May 2015, Accepted: 10 May 2015
Published online: 1 Sep. 2015

Abstract: The divisible load scheduling is a paradigm in the area of distributed computing. The traditional divisible load theory is based on the fact that, the communications and computations are obedient and do not cheat the algorithm. The literature of review shows that the divisible load model fail to achieve its optimal performance, if the processors do not report their true computation rates. The divisible load scheduling with uncertain communication rates has not been considered in the existing research. This problem lead us to propose a priority based divisible load scheduling method. The goal is to decrease the negative effects of communication rate cheating on the total finish time. The proposed method has been examined on several function approximation problems. It is found that the proposed method is extremely more efficient than either of the other methods.

Keywords: Divisible load scheduling, priority-based method, communication rate cheating, analytical hierarchy process (AHP)

1 Introduction

The first articles which concerned the divisible load theory (DLT) were published in 1988, [1,2]. Based on the DLT, it is assumed that the computation and communication can be partitioned into some arbitrary sizes, in which each part can be processed independently by a processor.

Over the past two decades, the DLT has found a wide variety of applications in the area of parallel processing, e.g., linear algebra [3], image and vision processing [4,5, 6], and data grid applications [7]. Moreover, the DLT was applied to a wide variety of interconnection topologies, including daisy chain, bus, single-level tree , multi-level tree [8], three-dimensional meshes [9], *k*-dimensional mesh [10], hypercubes [11], and arbitrary graphs [12]. It also has been applied in heterogeneous [13], homogeneous platforms [14], grid based method scheduling [7,15], and cloud based job scheduling [16]. There are extensive recent studies concerning the various aspects of divisible load scheduling theory, including, multi-installment processing [17], adaptive and probing strategies [18,19,20], memory limitation [21], and so on. A comprehensive review on the divisible load scheduling can be found in [22].

The traditional DLT is based on the fact that, the processors report their true computation and communication rates, i.e., they do not cheat the algorithm. In the real applications, the processors may cheat the algorithm. It means, the processors may not report their true computation or communication rates. This issue was investigated by Thomas E. Carroll and Daniel Grosu in their research publications [23,24]. The results of their research indicate that the computation cheating reduces the performance of the divisible load scheduling. In fact, the DLT obtains its optimal performance only if the processors report their true computation rates. The same problem can be considered in the communication rate as well. It means, the communication rate cheating also may decreases the performance of computing in the divisible load scheduling model. This paper focuses on the communication rate cheating problem. In order to reduce the effects of communication rate cheating problem on the performance of divisible load scheduling, we propose a priority-based divisible load scheduling method. The priority-based method is a new approach in the area of the divisible load scheduling.

* Corresponding author e-mail: myrshg@gmail.com, mothman@upm.edu.my

The rest of this paper is organized as follows. Section 2 gives some information concerning the background and related works. Section 3 is the preliminaries of this paper. Section 4 explains the proposed method and related algorithms, and section 5 presents some experimental results to support the proposed method. Finally, section 6 provides a conclusion.

2 Background

2.1 Divisible Load Scheduling

In general, the DLT assumes that, the computation and communication can be divided into some parts of arbitrary sizes, and these parts can be independently processed in parallel. The DLT assumes that, initially amount V of load is held by the originator p_0. A common assumption is that, the originator does not conduct any computation. It only distributes the load into parts α_0, α_1, …, α_m to be processed on worker processors p_0, p_1, …, p_m. The condition for the optimal solution is that, the processors stop processing at the same time; otherwise, the load could be transferred from busy to idle processors to improve the solution time [25]. The goal is to calculate α_0, α_1, …, α_m in the DLT timing equation. According to [26] the timing equation (close form) for a single-level tree network can be written by the following equations:

$$\alpha_j = \left(\frac{z_{j-1}T_{cm} + w_{j-1}T_{cp}}{z_j T_{cm} + w_j T_{cp}}\right)\alpha_{j-1} \tag{1}$$

and

$$\alpha_0 = \left(\frac{z_1 T_{cm} + w_1 T_{cp}}{w_1 T_{cp}}\right)\alpha_1 \tag{2}$$

Moreover, the total finish time can be calculated by the following equation:

$$T = \alpha_0 w_0 T_{cp} \tag{3}$$

where $\alpha_0 + \alpha_1 + \cdots + \alpha_m = V$. Throughout the paper we assume that $T_{cp} = T_{cm} = 1$.

The Gantt chart-like diagram for this case is depicted in Fig. 1.

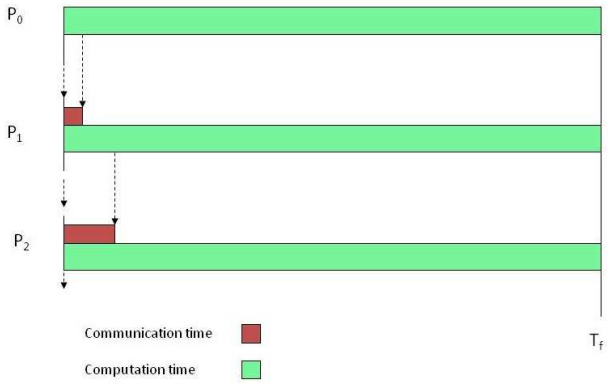

Fig. 1: Gantt chart-like timing diagram for divisible load in single level tree network.

2.2 Analytical Hierarchy Process

The first article, concerning the analytical hierarchy process (AHP) was published in [27]. It is a multi-criteria decision-making (MCDM)/ multi-attribute decision-making (MADM) model. Over the past two decades, the AHP has found a number of applications in various fields [28,29]. The AHP is a suitable method for solving priority-based scheduling with a wide range of attributes and alternatives as well [30]. In general, the AHP consists of three levels, including objective level, attributes level, and alternatives level. Each level uses comparison matrices for comparing the priorities.

Assume that $A = [a_{ij}]$ is a comparison matrix. Each entry in matrix A is positive. In this case, A is a square matrix ($A_{n\times n}$). There is only a vector of weights such as u=(u_1, u_2, …, u_n) associated with any arbitrary comparison matrix such as A. The relationship between the elements of comparison matrix (A) and its vector of weights (u) is shown in the following equation:

$$a_{ij} = \begin{cases} \frac{u_i}{u_j} & i \neq j \\ 1 & i = j \end{cases} \tag{4}$$

An essential step in AHP is to calculate vector of weights(u) which can be computed by the following equation:

$$Au = \lambda_{max}.u \tag{5}$$

where λ_{max} denotes the principal eigenvalue of A and u denotes the corresponding eigenvector. If A is absolutely consistent, then $\lambda_{max} = n$.

A metric for evaluating consistency of comparison matrix is named consistency rate (CR), it can be calculated by the following equation:

$$CR = \frac{CI}{RI} \tag{6}$$

where RI and CI denote the random index and consistency index respectively. The consistency index (CI) can be calculated as the following equation:

$$CI = \frac{\lambda_{max} - n}{n-1} \tag{7}$$

If $CR < 0.1$ then comparison matrix will be consistent. Furthermore, RI in Eq. (6) can be obtained by using Table 1. Other methods for calculating RI are available in [27,29,31].

Table 1: Random Index (*RI*) vs. the number of rows (N) of matrix

N	2	3	4	5	6	7	8	9
RI	0.00	0.58	0.90	1.12	1.24	1.32	1.41	1.45

2.3 Related Works

The main idea of the processor cheating refers to misreporting and time varying problem which was investigated in respect of divisible load scheduling in 1998 [32]. A few years later, Thomas E. Carroll *et al.*, focused on application case of misreporting in the divisible load scheduling [23,24]. They proposed a strategyproof mechanism for the divisible load scheduling under various topologies, including the bus and multi-level tree network. However, the cheating problem may occur if the processors execute their fraction of loads with different rates. Suppose that the root processor allocates α =(α_0, α_1, …, α_m) fraction of load for processors. This allocation is based on the assumption that, the computation and communication rates of p_j (j = 1, 2, …, m) are equal to w_j and z_j respectively. In fact, p_0 learns the actual computation rate of p_j once p_j completes execution of its fraction of load. The root processor also learns the actual communication rate once the fraction of load is sent to the worker processors and received the response. Carroll *et al.*, indicated that the divisible load scheduling model obtains its optimal performance only if the processors report their true computation rates. Subsequently, the problem was continued by the other researchers [33]. In [33] a multi-objective divisible load method has been proposed. The multi-objective method can reduce the effects of computation rate cheating on the performance of the divisible load scheduling. The same problem concerning the communication rate cheating can be considered. This paper focuses on the communication rate cheating on the performance of divisible load scheduling. For this purpose we use the analytical hierarchy process (AHP). The first application of the AHP concerning the DLT was proposed in [34]. That work contains a general form of a multi-criteria divisible load scheduling. In the present study we propose a priority-based divisible load method. The proposed method is able to handle the priority of processors in order to reduce the effects of communication rate cheating on the performance.

3 Preliminaries

Table 2 indicates the basic notation used in this paper.

Definition 1.*(Ranking Function). Assume that T_1 and T_2 are two real numbers. For any $s \in R^+$ (positive real number), we find n that satisfies the following equation:*

$$\left\lceil \frac{T_1 - T_2}{s} \right\rceil = n \tag{8}$$

Table 2: Definitions and Notations.

Notation	Description	Notation	Description
p_j	The j^{th} processor	m	Number of processors
w_j	Computation rate of p_j	z_j	Communication rate of p_j
V	Total size of data	v	Size of probing
α_j	Initial fraction of load for p_j	k	Number of probing
T_i^e	Expected finish time	T_i^o	Observed finish time in the i^{th} probing
$\hat{T}_{ij}^o$	Observed time of i^{th} probing for p_j	$\hat{z}_{ij}$	Communication rate in i^{th} probing for p_j
T(V)	Time taken for processing V	$T(v)$	Time taken for processing v

A ranking function denoted by $\Psi(T_1, T_2)$ can be defined as the following equation:

$$\Psi(T_1, T_2) = \begin{cases} n+1 & n > 0 \\ \frac{1}{1-n} & n < 0 \\ 1 & n = 0 \end{cases} \tag{9}$$

Lemma 1.*Assume that T_1 and T_2 are two real numbers and Ψ is a ranking function. The following equation indicates the main property of Ψ:*

$$\Psi(T_1, T_2) = \frac{1}{\Psi(T_2, T_1)} \tag{10}$$

Lemma 2.*Assume that A is a comparison matrix, and $e = (1, 1, \ldots, 1)^T$. We also assume that u and λ_{max} are principal eigenvector and eigenvalue of A respectively. Principal eigenvector (u) can be calculated as the following equation:*

$$u = \lim_{\ell \to \infty} \frac{1}{\ell} \sum_{t=1}^{\ell} \frac{A^t e}{e^T A^t e} \tag{11}$$

also λ_{max}, the corresponding eigenvalue of u can be calculated as the following equation:

$$\lambda_{max} = \lim_{\ell \to \infty} \frac{e^T A^{\ell+1} e}{e^T A^{\ell} e} \tag{12}$$

*Proof.*A method for proof can be found in [27].

Lemma 3.*Assume that $A_{n \times n} = (a_{rs})$ is a comparison matrix, then A is consistent if:*

$$a_{rs} = a_{rq} \times a_{sq} \tag{13}$$

for all r, s, q=1, 2, . . . , n [27].

4 Proposed Model

4.1 Problem Description

The basic assumption for the DLT is that, the communications and computation rates are obedient. This assumption can be provided by the following theorem [35].

Theorem 1. *Assume that $p_1, p_2, \ldots, p_m$ are m processors which are interconnected to p_0, by a single-level tree network topology. We also assumed that $z_0, z_1, \ldots, z_m$ are the communication rates of the m processors. It is also assumed that $w_0, w_1, \ldots, w_m$ are the computation rates of the $m+1$ processors. Thus, the processing time will be minimum if the processors satisfy the two following conditions:*

1. $z_j \leq z_{j+1}$
2. $w_j \leq w_{j+1}$.

In this paper we assume that the computation rates are obedient. We investigates the divisible load scheduling in a different priority. The following theorems indicate the motivation of this paper.

Theorem 2. *Considering the assumptions of Theorem 1, the communication rate cheating problem increase the processing time.*

Proof. Clearly the communication rate cheating problems change the optimal arrangement of processors which was explained in Theorem 1. Therefore the communication rate cheating problems increase the processing time in a divisible load scheduling model.

The proposed priority-based method uses a communication-based probing strategy. The probing strategies are applied k times on the processors. The goal is to provide a set of communication rates for the worker processors. In fact, we obtain k different priorities for the processors. By using the analytical hierarchy process (AHP) the priorities can be combined. The following theorem indicates how to combine the priorities in order to estimate the best priority for the processors.

Theorem 3. *Assume that $\rho^i = (\rho_1^i, \rho_2^i, \ldots, \rho_j^i, \ldots, \rho_m^i)$ is a priority vector. It is also assumed that $\rho^1, \rho^2, \ldots, \rho^i, \ldots, \rho^k$ are k priority vectors and $r_1, r_2, \ldots, r_i, \ldots, r_k$ are corresponding priority values respectively. The best estimated priority can be computed by the following equation:*

$$\overline{\rho} = \begin{pmatrix} \rho_1^1 & \rho_1^2 & \cdots & \rho_1^k \\ \rho_2^1 & \rho_2^2 & \cdots & \rho_2^k \\ \vdots & \vdots & \vdots & \vdots \\ \rho_m^1 & \rho_m^2 & \cdots & \rho_m^k \end{pmatrix} \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_k \end{pmatrix} = \begin{pmatrix} \sum_{j=1}^{k} \rho_1^j r_j \\ \sum_{j=1}^{k} \rho_2^j r_j \\ \vdots \\ \sum_{j=1}^{k} \rho_m^j r_j \end{pmatrix} \quad (14)$$

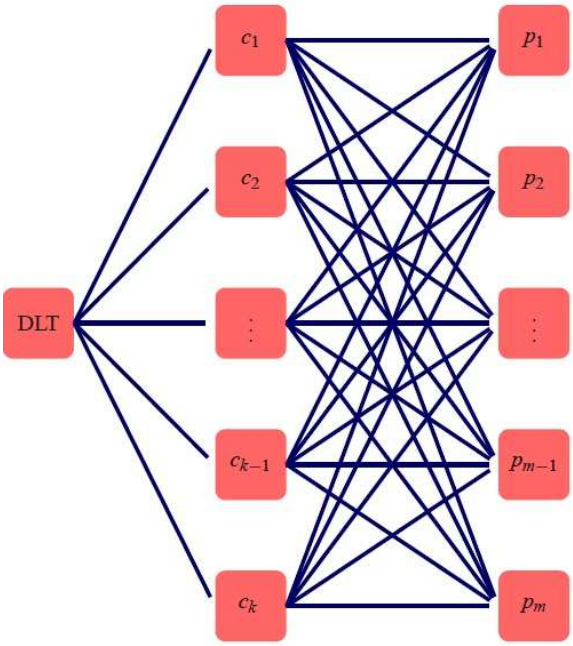

Fig. 2: A hierarchical framework for the proposed model.

4.2 Framework

The proposed method obtains some information about the communication rates of the worker processors. The gathered information helps the algorithm to calculate the estimated differences of communication rates of the worker processors. A hierarchical framework for the proposed method is shown in Fig. 2. In this framework, we consider three levels, including objective level (DLT), probing level ($c_1, c_2, \ldots, c_k$), and processor level ($p_1, p_2, \ldots, p_m$). At the first level we have only one node which presents the optimal value of scheduling. At the second level we consider k criteria. The criteria can be calculated in probing process. The total finish time in each probing process can be considered as the corresponding value of criterion. At the last level the processors are compared based on the total finish time. A descriptive framework for the proposed method has been shown in Fig. 3.

4.3 Description of the Proposed Method

The proposed method consists of three phases, including communication-based probing, decision making, and load allocation.

4.3.1 Communication-based Probing

In this phase, the originator obtains some information about the behavior of worker processors, including actual

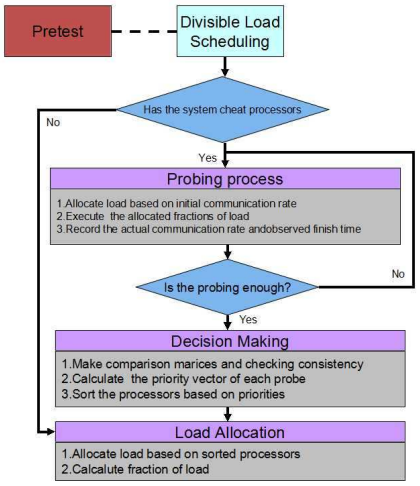

Fig. 3: A descriptive framework for the proposed model.

communication rates and cheated time. In fact, the originator learns the communication rates of worker processors after transferring the allocated load. The recorded information helps the originator to consider the best priority to the processors for allocating their fractions of load. This phase consists of the following steps:

–**step 1:** The originator distributes v $(v \ll V)$ amount of load on the worker processors. In this case each processor receives its fraction of load based on the initial communication rates of the worker processor.

–**step 2:** Each worker processor calculates its finish time based on its actual communication rate. It can be calculated by the following equation:

$$T_{ij}^{o} = \alpha_j \hat{z}_{ij} + \alpha_j w_j \tag{15}$$

–**step 3:** The root processor computes observed total finish time which is denoted by T_i^o. It can be calculated by the following equation:

$$T_i^o = Max(T_{i1}^o, T_{i2}^o, \ldots, T_{ij}^o, \ldots, T_{im}^o) \tag{16}$$

The originator also calculates the estimated finish time denoted by T_i^e.

–**step 4:** Information about the processors, including $\hat{z}_{ij}$ and $T_i^o - T_i^e$ would be recorded.

Table 3 indicates the population which is gathered by the communication-based probing phase. Furthermore, the details of probing phase is indicated in Algorithms 1.

Table 3: Sample of population produced by communication-based probing phase.

Probing	Cheated time	The changes of communication rates			
		p_1	p_2	$\ldots$	p_m
1	$T_1^o - T_1^e$	$\hat{z}_{11}$	$\hat{z}_{12}$	$\ldots$	$\hat{z}_{1m}$
2	$T_2^o - T_2^e$	$\hat{z}_{21}$	$\hat{z}_{22}$	$\ldots$	$\hat{z}_{2m}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ldots$	$\vdots$
$i-1$	$T_{i-1}^o - T_{i-1}^e$	$\hat{z}_{i-1,1}$	$\hat{z}_{i-1,2}$	$\ldots$	$\hat{z}_{i-1,m}$
i	$T_i^o - T_i^e$	$\hat{z}_{i1}$	$\hat{z}_{i2}$	$\ldots$	$\hat{z}_{im}$
$i+1$	$T_{i+1}^o - T_{i+1}^e$	$\hat{z}_{i+1,1}$	$\hat{z}_{i+1,2}$	$\ldots$	$\hat{z}_{i+1,m}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ldots$	$\vdots$
$k-1$	$T_{k-1}^o - T_{k-1}^e$	$\hat{z}_{k-1,1}$	$\hat{z}_{k-1,2}$	$\ldots$	$\hat{z}_{k-1,m}$
k	$T_k^o - T_k^e$	$\hat{z}_{k1}$	$\hat{z}_{k2}$	$\ldots$	$\hat{z}_{km}$

Algorithm 1 Communication-based Probing(w, z, v)

Input: A small part of load(v) and a network with $m+1$ processors
Output: A population of recorded data

Consider $Z = (z_1, z_2, \ldots, z_m)$.
Consider $Z_i = (z_{i1}, z_{i2}, \ldots, z_{im})$.
Distribute v load to processors based on Z_i using Algorithm 2.
Compute T_i^e
$i \leftarrow 1$
while $i \leq k$ **do**
  Distribute v load to processors based on $\hat{Z}_i$ using Algorithm 2.
  Calculate $\hat{Z}_i = (\hat{z}_{i1}, \hat{z}_{i2}, \ldots, \hat{z}_{im})$.
  Calculate the T_{ij}^o for $j = 1, 2, \ldots, m$; using Eq. (15).
  Calculate T_i^o using Eq. (16).
  Record $\hat{z}_{ij}$ for p_j
  Record $T_i^o - T_i^e$
  $i \leftarrow i+1$
end while
return: A population of recorded data.

Algorithm 2 Allocate(w, z, v)

Input: $\Phi = \{ p_0, p_1, \ldots, p_m \}$ a single level tree
Output: load allocating α to the processors

$j \leftarrow 1$
while $j \leq m$ **do**
  $k_j \leftarrow \frac{w_{j-1}}{z_j + w_j}$
end while
$\alpha_0 \leftarrow \frac{v}{1 + \Sigma_{j=1}^{m} \Pi_{s=1}^{j} k_s}$
$j \leftarrow 1$
while $j \leq m$ **do**
  $\alpha_j \leftarrow k_j \alpha_{j-1}$
end while

4.3.2 Decision Making

In this phase we use the gathered information of previous phase in order to estimate the best priority of processors. This phase consists of four following steps:

–*Step 1: (Making Comparison Matrix for Criteria).* In this step, each of the probing process can be considered as a criterion. The comparison matrix of communication criterion can be calculated by Algorithm 3. The algorithm uses the difference of observed finish time (T_i^o) and expected finish time (T_i^e) which are shown in Table 3.

Algorithm 3 Making Comparison Matrix for Criteria()

Input: $T_1^0 - T_1^e, T_2^0 - T_2^e, \ldots, T_k^0 - T_k^e$
Output: a comparison matrix for criteria

1: **for** r=1 to k **do**
2: **for** t=r+1 to k **do**
3: $C[r,t] = \Psi(T_r^0 - T_r^e, T_t^0 - T_t^e)$
4: $C[t,r] = \Psi(T_t^0 - T_t^e, T_r^0 - T_r^e)$
5: **end for**
6: **end for**
7: **for** r=1 to k **do**
8: C[r, r]=1
9: **end for**

–*Step 2: (Checking Consistency).* Each comparison matrix must be consistent. The consistency of produced comparison matrix in *step 1*, can be investigated by Algorithm 4.

Algorithm 4 Checking Consistency ()

Input: Comparison matrix $D_{n\times n}$
Output: Boolean
Description:

1: **Compute** λ_{max} **using** Eq. (12);
2: **Compute** CI **by solving** Eq. (7);
3: **Compute** CR **by solving** Eq. (6);
4: **if** $CR \leq 0.1$ **then**
5: **matrix is consistent**
6: **else**
7: **matrix is not consistent**
8: **end if**

–*Step 3: (Making Comparison Matrix for Attributes).* In this step, comparison matrices must be computed for the processors based on the criteria. The comparison matrices present the effects of processor's cheating on the other processors. The comparison matrix of i^{th} probing can be defined as the following equation:

$$Q_{rs}^i = \begin{cases} \frac{\hat{z}_{is}}{\hat{z}_{ir}} & r \neq s \\ 1 & r = s \end{cases} \quad (17)$$

Algorithm 5 Corresponding Principal Eigenvector ()

Input: A Comparison matrix $D_{n\times n}$
Output: Corresponding principal eigenvector of $D_{n\times n}$

1: $\ell \leftarrow 1$
2: $e \leftarrow (1,1,\ldots,1)$
3: $h \leftarrow \frac{D^\ell e}{e^T D^\ell e}$
4: **while** (D is not consistent) **do**
5: $h \leftarrow h + \frac{D^\ell e}{e^T D^\ell e}$
6: $\ell \leftarrow \ell + 1$
7: **end while**
8: $\Lambda = \frac{h}{\ell}$
9: Return Λ

where Q^i is the comparison matrix of attributes in the i^{th} probing process. Using Lemma 3, it can be seen that Q^i is consistent.

–*Step 4: (Calculating the Vector of Weights).* Now, we have k consistent matrices. There are several methods for calculating the vector of weights of a consistent comparison matrix. The most frequent used method for calculating the vector of weights of comparison matrix is the *Sum-method* [27,28]. In this paper, we applied the *Sum-method*. Assume that Q^i is the i^{th} comparison matrix, then the corresponding vector of weights of Q^i can be calculated as the following equation:

$$u_r^i = \frac{\Sigma_{s=1}^m Q_{rs}^i}{\Sigma_{r=1}^m \Sigma_{s=1}^m Q_{rs}^i} \quad r = 1,2,\ldots,m; \quad i = 1,2,\ldots,k; \quad (18)$$

where k and m are the number of probing and the number of processors respectively.
We also suppose that Δ is defined as the following equation:

$$\Delta = [u^1 u^2 \ldots u^d] = \begin{pmatrix} u_1^1 & u_1^2 & \ldots & u_1^k \\ u_2^1 & u_2^2 & \ldots & u_2^k \\ \vdots & \vdots & \vdots & \vdots \\ u_m^1 & v_m^2 & \ldots & u_m^k \end{pmatrix} \quad (19)$$

Now, suppose that C is the comparison matrix of communication criteria. It can be calculated using Algorithm 3. we also calculate the priority vector of C using Algorithm 5. It is depicted by the following equation:

$$\Lambda = [\lambda_1 \lambda_2 \ldots \lambda_k]^T \quad (20)$$

Finally, we compute the *priority vector of distribution* denoted by PVD. It can be calculated by the following equation:

$$PVD = \Delta \times \Lambda \quad (21)$$

Algorithm 6 Decision Making ()

Input: m worker processors labeled by $p_1, p_2, \ldots, p_m$
Output: Allocated fraction of load
Description:

1: **Let** $i \leftarrow 1$
2: $\Delta \leftarrow$ nil
3: **while** $i \leq k$ **do**
4: **Make** Q^i using Eq. (17).
5: **Calculate** u^i for Q^i using Eq. (18)
6: **Attach** u^i to Δ
7: $i \leftarrow i+1$
8: **end while**; **Note:** at the end of this loop we have $\Delta = [u^1 u^2 \ldots u^k]$
9: **Make** C as the comparison matrix of criteria based on probing using Algorithm 3.
10: **Calculate** Λ for R using Algorithm 5.
11: $PVD \leftarrow \Delta \times \Lambda$;
12: **for** $i = 1$ to m **do**
13: **Assign** processor P_j with j^{th} element of PVD
14: **end for**
15: **Sort** processors based on their PVD value
16: **Allocate** fraction of load to the sorted processors using Algorithm 2.

The PVD can be also shown as the following equation:

$$PVD = \begin{pmatrix} u_1^1 & u_1^2 & \ldots & u_1^k \\ u_2^1 & u_2^2 & \ldots & u_2^k \\ \vdots & \vdots & \vdots & \vdots \\ u_m^1 & u_m^2 & \ldots & u_m^k \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_k \end{pmatrix} = \begin{pmatrix} \Sigma_{j=1}^k u_1^j \lambda_j \\ \Sigma_{j=1}^k u_2^j \lambda_j \\ \vdots \\ \Sigma_{j=1}^k u_m^j \lambda_j \end{pmatrix} \quad (22)$$

Each element of PVD is the corresponding priority value of a processor to get a fraction of load.

4.3.3 Load Allocation

In this phase the worker processors are sorted based on their corresponding value of the PVD. The root processor allocates the fraction of load to the sorted processors. A processor with the highest PVD value must obtain its fraction of load first. The details of proposed method are indicated by Algorithm 6.

4.4 Complexity Analysis

In this section, we calculate the effects of complexity on the total finish time in the proposed method. In this case the operational definition of complexity of proposed method is the number of computations that the root processor must execute along with the total finish time. The complexity of computation in the proposed method can be calculated by the following equation:

$$t_{complexity} = t_{probing} + t_{calculating} \quad (23)$$

Clearly, $t_{probing}$ is the complexity of Algorithm 1 . It can be calculated by following equation:

$$t_{probing} = c_1 \times k \times max(\alpha_j \hat{z}_{ij}) \quad (24)$$

Moreover, $t_{calculating}$ is the complexity of Algorithm 6. It can be calculated by following equation:

$$t_{calculating} = c_2 \times \ell \times k^2.81 + c_3 \times \ell \times m^{2.81} \quad (25)$$

where c_1, c_2, and c_3 are three constant numbers. The other parameters, including k, ℓ, and m are the number of probing, the number of steps to obtain a consistent comparison matrix, and the number of processors respectively. In Eq. (25), we consider Strassen's algorithm [36] for multiplication of two matrices.

5 Experimental Results

The experimental result consists of the three following scenarios:

– **Scenario 1**: In the first scenario, we consider five processors, which are interconnected in a single level tree network, see Fig. 4. It is assumed that, p_0 is the root processor and p_1, p_2, p_3 and p_4 are the worker processors. It is also assumed that w_0=0.1, z_0=0, w_1=0.1, z_1=0.01, w_2=0.2, z_2=0.02, w_3=0.3, z_3=0.03, w_4=0.4, and z_4=0.04. We perform the probing process three times. In this case, it is assumed that the four worker processors do not cheat the algorithm in the three probing processes.

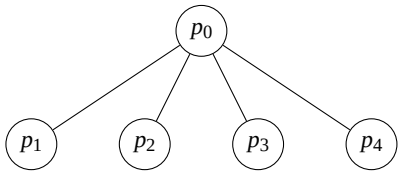

Fig. 4: Single level tree network with five processors.

Thus the comparison matrix for the probing processes can be shown by the following equation:

$$R = \begin{pmatrix} 1.00 & 1.00 & 1.00 \\ 1.00 & 1.00 & 1.00 \\ 1.00 & 1.00 & 1.00 \end{pmatrix}$$

The vector of weights for matrix R, can be calculated as follows:

$$\Lambda = \begin{pmatrix} 0.33 \\ 0.33 \\ 0.33 \end{pmatrix}$$

We also calculate the comparison matrix for processors in each probing process. In the three probing processes we have the following matrices:

$$Q^{Pr_1} = Q^{Pr_2} = Q^{Pr_3} = \begin{pmatrix} 1.00 & 2.00 & 3.00 & 4.00 \\ 0.50 & 1.00 & 1.50 & 2.00 \\ 0.33 & 0.66 & 1.00 & 1.33 \\ 0.25 & 0.50 & 0.75 & 1.00 \end{pmatrix}$$

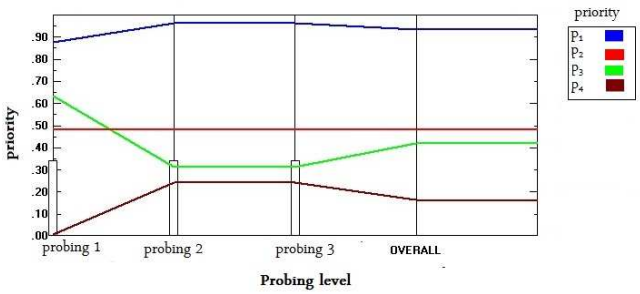

Fig. 5: The priority of processors in three probing.

Now, we calculate Δ using Eqs. (18) and (19). Thus, we have:

$$\Delta = [u^1 u^2 u^3] = \begin{pmatrix} 0.40 & 0.40 & 0.40 \\ 0.20 & 0.20 & 0.20 \\ 0.13 & 0.13 & 0.13 \\ 0.10 & 0.10 & 0.10 \end{pmatrix}$$

Finally, we calculate the PVD as the following matrix:

$$PVD = \Delta.\Lambda = \begin{pmatrix} 0.40 & 0.40 & 0.40 \\ 0.20 & 0.20 & 0.20 \\ 0.13 & 0.13 & 0.13 \\ 0.10 & 0.10 & 0.10 \end{pmatrix} \begin{pmatrix} 0.33 \\ 0.33 \\ 0.33 \end{pmatrix} = \begin{pmatrix} 0.40 \\ 0.20 \\ 0.13 \\ 0.10 \end{pmatrix}$$

According to the proposed algorithm, at first, the processor p_1 gets its fraction of load, because it has the highest value of PVD which is equal to 0.40. Subsequently, p_2, p_3, and p_4 receive their fraction of load respectively. Hence, we have:
α_0 = 0.32634, α_1 = 0.32634, α_2 = 0.16154, α_3 =0.10662, α_4 = 0.07917 and makespan=0.003263. This case indicates that, if the processors do not cheat the algorithm, then the proposed method works the same as the traditional divisible load models. Fig. 5 shows the priority of processors to obtain the fraction of load.

– **Scenario 2**: In the second scenario, we also consider five processors interconnected in a single level topology. The first processor (p_0) is the root processor and p_1, p_2, p_3 and p_4 are worker processors. It is assumed that $w_i = 0.1 \times i$ and $z_i = 0.01 \times i$, for $i = 0,1,2,3,4$. The probing process has been applied four times. It is also assumed that z_i can be changed in the various probing processes. The processing time in the four probing is indicated in Fig. 6. The details of probing process are shown in Table 4.

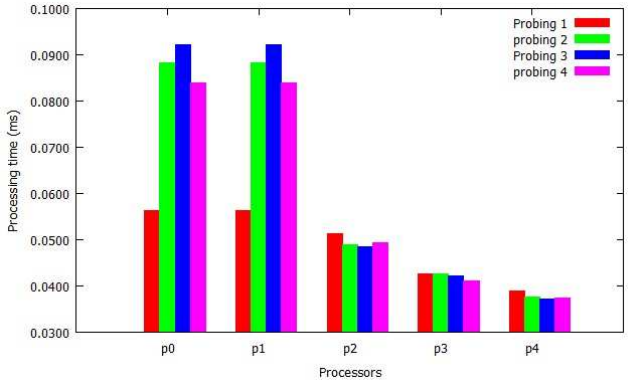

Fig. 6: Processing time in different probing processes.

Table 4: Sample of population produced by probing phase.

Probing	Cheated time	Communication rates			
		p_1	p_2	p_3	p_4
1	0.00582	0.02	0.03	0.04	0.01
2	0.03394	0.03	0.04	0.02	0.01
3	0.03803	0.05	0.06	0.07	0.09
4	0.03178	0.08	0.01	0.07	0.09

Using Algorithm 3, the comparison matrix for probing can be calculated as follows:

$$C = \begin{pmatrix} 1.000 & 35.561 & 31.046 & 3.852 \\ 0.028 & 1.000 & 247.500 & 0.002 \\ 0.032 & 0.004 & 1.000 & 0.006 \\ 0.269 & 462.963 & 160.000 & 1.000 \end{pmatrix}$$

Using Algorithm 5, the corresponding principal eigenvector of C can be calculated as follows:

$$\Lambda = \begin{pmatrix} 0.663 \\ 0.114 \\ 0.102 \\ 0.121 \end{pmatrix}$$

Moreover, the comparison matrices of processors are shown in Tables 5-8.
Therefore, the PVD for priority of processors can be calculated as follows:

$$PVD = \Delta.\Lambda = \begin{pmatrix} 0.159 & 0.090 & 0.132 & 0.480 \\ 0.120 & 0.725 & 0.110 & 0.240 \\ 0.240 & 0.103 & 0.662 & 0.160 \\ 0.480 & 0.080 & 0.142 & 0.120 \end{pmatrix} \begin{pmatrix} 0.663 \\ 0.114 \\ 0.102 \\ 0.121 \end{pmatrix}$$

Table 5: Comparison matrices of processors for communication-based probing 1.

Processors	*Processors*				u^1
	p_1	p_2	p_3	p_4	
p_1	1.000	1.333	0.666	0.333	0.159
p_2	0.750	1.000	0.500	0.250	0.120
p_3	0.333	0.666	1.000	1.333	0.240
p_4	3.000	4.000	2.000	1.000	0.480

Table 6: Comparison matrices of processors for communication-based probing 2.

Processors	*Processors*				u^2
	p_1	p_2	p_3	p_4	
p_1	1.000	1.125	0.875	1.125	0.090
p_2	8.000	1.000	7.000	9.000	0.725
p_3	1.142	0.142	1.000	1.285	0.103
p_4	0.888	0.111	0.777	1.000	0.080

Table 7: Comparison matrices of processors for communication-based probing 3.

Processors	*Processors*				u^3
	p_1	p_2	p_3	p_4	
p_1	1.000	1.200	0.200	1.400	0.132
p_2	0.833	1.000	0.166	1.166	0.110
p_3	5.000	6.000	1.000	7.000	0.662
p_4	0.714	0.857	1.142	1.000	0.142

Table 8: Comparison matrices of processors for communication-based probing 4.

Processors	*Processors*				u^4
	p_1	p_2	p_3	p_4	
p_1	1.000	2.000	3.000	4.000	0.480
p_2	0.500	1.000	1.500	2.000	0.240
p_3	0.333	0.666	1.000	1.333	0.160
p_4	0.250	0.500	0..750	1.000	0.120

Lastly, *priority vector of distribution* is:

$$PVD = \Delta.\Lambda = \begin{pmatrix} 0.218 \\ 0.338 \\ 0.312 \\ 0.132 \end{pmatrix}$$

The performance analysis of the second scenario has been shown in Fig. 7. The figure puts the information about how alternatives behave on each criterion. Each criterion possesses a vertical line. The overall priority of each alternative is where it intersects the axis on the right. The priority of each criterion is shown by the rectangular box on that criterion's vertical line, as read from the axis at the left. As the priority changes, the overall priorities of the alternatives on the axis at the right change. The priorities of processors based on communication criteria are shown in Fig. 8.

Therefore, at first the processor p_2 obtains its fraction of load. Subsequently, processors p_3, p_1, and p_4 receive their fraction of loads respectively.

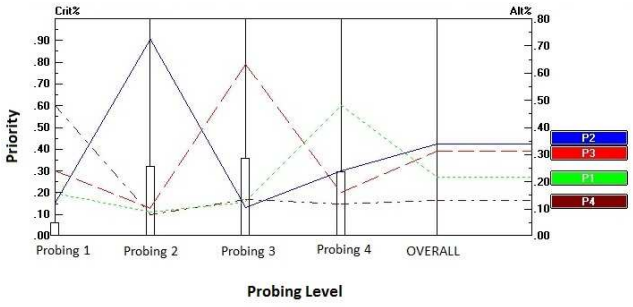

Fig. 7: Performance analysis of the second scenario.

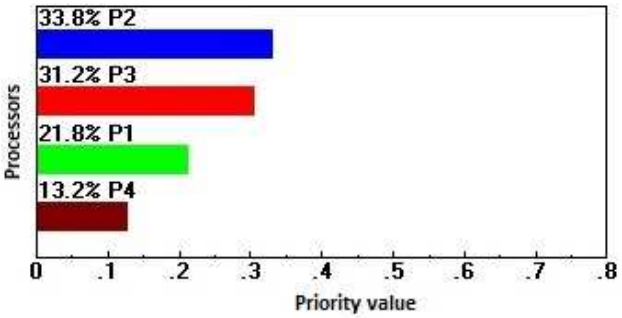

Fig. 8: The priority of processors based on communication criteria.

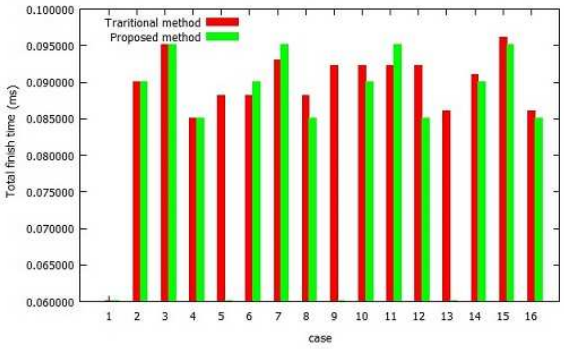

Fig. 9: Evaluation of proposed method in 16 different cases.

– **Scenario 3**: In the third scenario, we investigate the effects of communication rate cheating on the processing time. We simulate the proposed method with a set of random value of communication rate cheating. We consider the five previous processors and assumed that $w_0 = 1$, $w_i = 0.1 \times i$ for $i = 1,2,3,4$. The communication rate cheating for each processor is generated using the Poisson distribution with mean value of 0.05 ($\overline{z}_i = 0.05$ for i =1, 2, 3, 4 and $z_0 = 0$). The test has been done more than 500 times. We compare the average processing time which are calculated by the proposed method and the traditional divisible load. The result has been shown in Fig. 9. As the figure shows the proposed method has the lowest total finish time in almost all cases comparing to the traditional method. Furthermore, the total finish time in the four cases of the proposed method, including 1, 5, 9, and 13 dramatically have been decreased.

6 Conclusion

Existing divisible load scheduling algorithms are based on the fact that, the processors report their true communication rate to the root processor. Based on this assumption the traditional divisible load considers that $z_j \leq z_{j+1}$ for all processors. In this paper we assumed that the actual communication rate for sending fraction of load to the processors may be different from the true communication rates. In this case, the priority of the processors for obtaining the fraction of loads would be changed. Therefore, we proposed a priority-based divisible load scheduling method for the first time. The proposed method is able to estimate the actual communication rates of the processors. The experimental results indicated that the proposed method is able to reduce the effects of communication rate cheating on the performance of divisible load scheduling. We also calculated the complexity of computation and indicated that the complexity of computation of proposed method is ignorable. In this work, we calculated the priority of processors only based on communication rates.

In the future, we will develop the priority of processors based on both communication and computation rates of the processors.

Acknowledgments

This work has been supported by the Malaysian Ministry of Education under the Fundamental Research Grant Scheme FRGS/1/11/SG/UPM/01/1.

References

[1] Yuan-Chieh Cheng and Thomas G. Robertazzi. Distributed computation with communication delay. IEEE Transactions on Aerospace and Electronic Systems 24(6):700-712, 1988.

[2] Rakesh Agrawal and H.V. Jagadish. Partitioning techniques for large-grained parallelism. IEEE Transactions on Computers 37(12):1627-1634, 1988.

[3] SK Chan and Bharadwaj Veeravalli and Debasish Ghose. Large matrix-vector products on distributed bus networks with communication delays using the divisible load paradigm: performance analysis and simulation. Mathematics and Computers in Simulation 58(1):71-92, 2001.

[4] Bharadwaj Veeravalli, Xiaolin Li, and Chi Chung Ko. Efficient partitioning and scheduling of computer vision and image processing data on bus networks using divisible load analysis. Image and Vision Computing 18(11):919-938, 2000.

[5] Lee Chi-kin and Hamdi Mounir. Parallel image processing applications on a network of workstations. Parallel Computing 21(1):137-160, 1995.

[6] Ping Li and Bharadwaj Veeravalli and Ashraf A. Kassim. Design and implementation of parallel video encoding strategies using divisible load analysis. IEEE Transactions on Circuits and Systems for Video Technology 15(9):1098-1112,2005.

[7] Monir Abdullah, Mohamed Othman, Hamidah Ibrahim, and Shamala Subramaniam. Optimal workload allocation model for scheduling divisible data grid applications, author. Future Generation Computer Systems 26(7):971-978:2010.

[8] Thomas G. Robertazzi. Divisible Load Modeling for Grids. Springer New York, 2007.

[9] Maciej Drozdowski and Włodzimierz Głazek. Scheduling divisible loads in a three-dimensional mesh of processors. Parallel Computing 25(4):381-404, 1999.

[10] Chang Yeim-Kuan and Wu Jia-Hwa and Chen Chi-Yeh and Chu Chih-Ping. Improved methods for divisible load distribution on k-dimensional meshes using multi-installment. IEEE Transactions on Parallel and Distributed Systems 18(11):1618-1629, 2007.

[11] Jacek Bazewicz and Maciej Drozdowski. Scheduling divisible jobs on hypercubes. Parallel computing 21(12):1945-1956, 1995.

[12] Yao Jingnan and Bharadwaj Veeravalli. Design and performance analysis of divisible load scheduling strategies on arbitrary graphs. Cluster Computing 7(2):191-207, 2004.

[13] Beaumont Olivier and Legrand Arnaud and Robert Yves. Scheduling divisible workloads on heterogeneous platforms. Parallel Computing 29(9):1121-1152, 2003.

[14] Jacek Błażewicz and Maciej Drozdowski and Mariusz Markiewicz.Divisible task scheduling: concept and verification. Parallel Computing 25(1):87-98, 1999.

[15] Vladimir V. Korkhov and Jakub T. Moscicki and Valeria V. Krzhizhanovskaya. The user-level scheduling of divisible load parallel applications with resource selection and adaptive workload balancing on the grid. Systems Journal IEEE 3(1):121-130, 2009.

[16] Monir Abdullah and Mohamed Othman. Cost-based multi-QoS job scheduling using divisible load theory in cloud computing. Procedia Computer Science 18:928-935, 2013.

[17] Amin Shokripour, Mohamed Othman, Hamidah Ibrahim, and Shamala Subramaniam. New method for scheduling heterogeneous multi-installment systems. Future Generation Computer Systems, 28(8):1205-1216, 2012.

[18] Ghose Debasish. A feedback strategy for load allocation in workstation clusters with unknown network resource capabilities using the DLT paradigm. Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications 1:425-428, 2002.
[19] Debasish Ghose and Kim Hyoung Joong and Kim Tae Hoon. Adaptive divisible load scheduling strategies for workstation clusters with unknown network resources. IEEE Transactions on Parallel and Distributed Systems 16(10):897-907, 2005.
[20] Kim Hyoung Joong. A novel optimal load distribution algorithm for divisible loads. Cluster Computing 6(1):41-46, 2003.
[21] Berlińska, J and Maciej Drozdowski. Heuristics for multi-round divisible loads scheduling with limited memory. Parallel Computing 36(4):199-211, 2010.
[22] Shamsollah Ghanbari and Mohamed Othman. Comprehensive Review on Divisible Load Theory: Concepts, Strategies, and Approaches. Mathematical Problems in Engineering, 2014. http://dx.doi.org/10.1155/2014/460354
[23] Thomas E. Carroll and Daniel Grosu. An incentive-based distributed mechanism for scheduling divisible loads in tree-networks. J.ParallelDistrib.Comput 72:389-401, 2012.
[24] Thomas E. Carroll and Daniel Grosu. Strategyproof mechanisms for scheduling divisible loads in bus-networked distributed systems. IEEE Transactions on Parallel and Distributed Systems 19(8:1124-1135,2008.
[25] Jeeho Sohn and Thomas G. Robertazzi. Optimal load sharing for a divisible job on a bus network. Proceedings of the 1993 Conference on Information Sciences and Systems, pp. 835-840, 1993.
[26] Thomas G. Robertazzi. Ten reasons to use divisible load theory. Computer 36(5):63-68,2003.
[27] Thomas L. Saaty. What is the analytic hierarchy process. Springer. '1988.
[28] Thomas L. Saaty. How to make a decision: the analytic hierarchy process. European journal of operational research 48(1):9-26, 1990.
[29] Thomas L. Saaty. The modern science of multicriteria decision making and its practical applications: the AHP/ANP approach. Operations Research 6(15):1101-1118, 2013.
[30] Shamsollah Ghanbari and Mohamed Othman. A priority based job scheduling algorithm in cloud computing. Procedia Engineering 50:778-785, 2012.
[31] Patrick S. Chen and Peter Chu and Michelle Lin. On vargas's proof of consistency Test for 3x3 comparison matrices in AHP. Journal of the Operations Research Society of Japan-Keiei Kagaku 45(3):233-242, 2002.
[32] Sohn Jeeho and Thomas G. Robertazzi. Optimal time-varying load sharing for divisible loads. IEEE Transactions on Aerospace and Electronic Systems 34(3):907-923, 1998.
[33] Shamsollah Ghanbari, Mohamed Othman, Mohd Rizam Abu Bakar, and Wah June Leong. Multi-objective method for divisible load scheduling in multi-level tree network. Future Generation Computer Systems (2015), In Press. http://dx.doi.org/10.1016/j.future.2015.03.015
[34] Shamsollah Ghanbari, Mohamed Othman, Wah June Leong, and Mohd Rizam Abu Bakar. Multi-criteria based algorithm for scheduling divisible load. Lecture Notes in Electrical Engineering (LNEE) 1:547-554, 2014.
[35] BharadwajVeeravalli and Debasish Ghose and Venkataraman Mani and Tomas G. Robertazz. Scheduling divisible loads in parallel and distributed systems, IEEE, 1996.
[36] Volker Strassen. Gaussian elimination is not optimal. Numerische Mathematik 13(4):354-356, 1969.

Shamsollah Ghanbari received his B.Sc. in the field of Applied Mathematics in Computer Science from Amir Kabir University, Tehran, Iran, 1997. He received his M.Sc. in applied mathematics from Mazandaran University, Iran, 2000. Currently, he is a PhD candidate in the Institute For Mathematical Research (INSPEM) of UPM (Universiti Putra Malaysia). His field study in Ph.D is high performance computing and distributed computing. Some of his current research interests are divisible load scheduling, grid computing, optimizations, parallel and distributed computing, complexity of algorithms. He has published articles in several refereed journals and international conferences.

Mohamed Othman received his PhD from the National University of Malaysia with distinction (Best PhD Thesis in 2000 awarded by Sime Darby Malaysia and Malaysian Mathematical Science Society). Now, he is a Professor in the Faculty of Computer Science and Information Technology, University Putra Malaysia (UPM). He is also an associate researcher at the Lab of Computational Science and Mathematical Physics, Institute of Mathematical Research (INSPEM), UPM. He published more than 320 National and International journals and proceeding papers. His main research interests are in the fields of parallel and distributed algorithms, network design and management (high-speed, security, wireless at PHY and MAC layers and traffic monitoring) and scientific computing.

Mohd Rizam Abu Bakar obtained his PhD in Applied Statistics from University of Bradford, United Kingdom. His research interest includes: Survival data analysis, data envelopment analysis and epidemic modelling. Currently he is an associate professor in the Department of Mathematics at Universiti Putra Malaysia (UPM). He is also an associate researcher of Institute for Mathematical Researches (INSPEM), UPM. Recently is head of the programme of Survival Analysis in Computational Statistics and Operations Research Laboratory of INSPEM.

Wah June Leong received his Ph.D. degree in Applied Mathematics from the University of Putra Malaysia in 2003. He is currently an academic staff at Faculty of Science, Universiti Putra Malaysia. His research interests include numerical optimization, control theory and stabilization.