Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®

Computer Communications 27 (2004) 679–693

computer communications

www.elsevier.com/locate/comcom

Design and analysis of a non-preemptive decentralized load balancing algorithm for multi-class jobs in distributed networks

Zeng Zeng, Bharadwaj Veeravalli*

Open Source Software Laboratory, Department of Electrical and Computer Engineering, The National University of Singapore, 4 Engineering Drive 3, Singapore, Singapore 117576

Received 18 April 2003; revised 4 February 2004; accepted 4 February 2004

Abstract

In this paper, we propose a *static*, *decentralized* load balancing algorithm for handling *multi-class* jobs in distributed network system for minimizing the mean response time of a job, using the concept of virtual routing. We formulate the problem as a *constrained non-linear minimization problem* with job flow-rate, communication delays, and processing delays, as constraints. We employ a novel approach to transform the formulated problem into an equivalent routing problem and propose an algorithm, referred to as load balancing via virtual routing (LBVR), to seek an optimal solution, whenever it exists. We show that the design of the proposed algorithm subsumes several interesting properties and guarantees to deliver a *super-linear* rate of convergence in obtaining an optimal solution, whenever it exists. Also, when the variation of mean link delays is assumed to be a convex function, we show that the solution generated by our LBVR algorithm is indeed an optimal solution, whereas, when the above variation is assumed to be non-convex, we derive a *necessary* condition for an optimal solution. With rigorous experiments we test our algorithm in terms of its rate of convergence and quality of solution to quantify its performance. We demonstrate the complete workings of our algorithm using an illustrative example in a systematic fashion, for ease of understanding.

Keywords: Static load balancing; Distributed systems; Communication delays; Mean response time; Routing; Non-preemptive scheduling

1. Introduction

Minimizing the mean response time (MRT) of the jobs submitted for processing in a parallel/distributed system is a critical performance metric to be considered for improving the overall performance of the system. Load balancing algorithms thrive to meet this objective of minimizing the MRT, the average time interval between the time instant at which a job is submitted and the time instant at which the job leaves the system after processing. The design of such load balancing algorithms, in general, considers several influencing factors. For instance, the underlying network topology, job arrival rates at each processor in the system, communication network bandwidth/traffic, etc. Further, while considering job characteristics, there may exist several variations, such as priority assignment for jobs in processing, jobs with or without deadlines, etc. In the exiting literature, several combinations of the above types and other issues such as sender-initiated strategies, receiver-initiated strategies are designed for load balancing [1].

Further, while balancing the loads, certain type of information such as the number of jobs waiting in the queue to be processed, the current job rates arriving, etc. at each processor as well as in neighboring processors may be exchanged among the processors for improving the overall performance. Based on the information that can be used, load balancing algorithms are classified as either *dynamic* or *static*. A dynamic algorithm makes its decision according to the state of the system, where the state could refer to the above-mentioned information [2,4,7,28]. On the other hand, a static algorithm is carried out by a predetermined policy, without considering the state of the system [3,5,8].

In this paper, we attempt to formulate a static load balancing algorithm as a non-linear constrained optimization problem. Specifically, we consider the following real-life situation in our problem setting/definition. We consider a network of processors to which several classes of jobs arrive with a constant flow-rate for processing. Each processor may

* Corresponding author. Tel.: +65-6874-5158; fax: +65-6779-1103.
E-mail addresses: elebv@nus.edu.sg (B. Veeravalli), engp1588@nus.edu.sg (Z. Zeng).

0140-3664/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2004.02.002

receive one or more classes of jobs and considers the entire set of jobs submitted for processing to it as its total input load. As with the principle of load balancing, jobs are allowed to migrate from heavily loaded processors to lightly load processors for minimizing the MRT. We assume that the underlying network has an arbitrary topology that incurs non-zero finite communication delays while transferring jobs between processors. Further, for each class of job, the communication delay is modeled as a non-linear function that depend on the network traffic, and that the delays are different on different links. Consequently, the nature of this function, either as a convex or a non-convex, influences the optimality of the solution. Also, we assume that each class of jobs demands different processing rate, depending on the nature of the jobs. All these influencing factors are captured as constraints in our optimization problem. Further, we consider a non-preemptive style of processing of the jobs at a processor, i.e. a job that is currently being processed cannot be interrupted by any other class of job for processing.

We now discuss some related literature to our problem context below and then point out to some key differences in the problem formulation, solution approaches, and the techniques used to solve the problem.

1.1. Related work

Numerous studies have been conducted on a variety of load balancing algorithms in the literature. In the following, we present some related studies that are very close to our contributions in this paper. An excellent compilation of most of the load balancing/sharing algorithms until 1992 can be found in Ref. [9]. For a dynamic load balancing algorithm, it is unacceptable to frequently exchange state information because of the high communication overheads. In order to reduce the communication overheads, Anand et al. [2] proposed an estimated load information scheduling algorithm (ELISA) and Michael [15] analyzed the usefulness of the extent to which old information can be used to estimate the state of the system. In Refs. [6,10,11], the authors have proven the correctness using randomization techniques, leading to an exponential improvement in balancing the loads. However, in Refs. [22,29], it was pointed out that the static load balancing algorithms are preferable when the system loads are moderate or light or when the communication overheads are still high.

Static load balancing algorithms are widely used in large-scale simulations [24], parallel program [25], etc. For static algorithms, there are some differences among the network configurations. Kim and Kameda [31] proposed two algorithms for static load balancing in star and bus network configurations, respectively. Also, for an optimal static load balancing, Li and Kameda [30] studied the tree network configurations with two-way traffic. In Refs. [5,8], the algorithms proposed were concerned about an arbitrary network configuration, and hence, became more applicable in a practical distributed system. However, the contributions mentioned above considered only a single class of jobs. In practice [32], the jobs in the system were divided into several classes and each class of jobs had its own priority. The study of multi-class jobs makes the system more flexible to handle different classes of jobs and is a right step in generalizing the study.

We discuss on some more related work later in Section 4.1, while discussing the correspondence between a load balancing problem and a routing problem.

1.2. Our contributions

Our contributions are summarized as follows. The problem addressed in this paper is closely related to the earlier works reported in Refs. [3,5,8], however, the key differences are as explained below. In Refs. [5,8], the formulated non-linear constrained optimization problem considers only one class of jobs and shows that the delay functions are indeed convex and increasing functions. Whereas, in Ref. [3] the delay functions are assumed to be convex and the proposed algorithm is proven to be faster than the standard flow deviation (FD) algorithm [21], consuming larger amount of computations in carrying out certain inverse functions. Also, the formulated problem in this work considers the process of load distribution in a different manner. For instance, for each class of jobs and for each processor, say i, the neighboring processors are categorized into four different sets such that, processors in each set send the jobs of this class to a processor i based on certain rules. The rationale for this may be driven from application needs.

In our formulation, we relax this assumption and we consider all the jobs of a class that arrive at node i as a cumulative amount regardless of their origin. Thus, in our model, each processor is considered as an unbiased resource capable of processing the submitted jobs. Also, the delay functions are considered as arbitrary non-linear functions for the analysis to be more generic. Of course, as a possible extension, we also analyze the performance when convexities of the delay functions are to be considered. As a solution approach, we propose a novel methodology for the posed problem. We transform the problem into a routing problem and derive an optimal solution to the transformed problem. The correspondence between the load balancing problem and the routing problem is also discussed. Significant advantages of our proposed algorithm are discussed in Section 5. Thus, in this paper, we propose a *static*, *decentralized* load balancing algorithm for *multi-class* jobs in distributed network systems for minimizing the MRT of a job, using the concept of virtual routing.

The organization of the paper is as follows. In Section 2, we present a mathematical model and the problem definition. In Section 3, we formulate the problem and discuss on the solution approach. In Section 4, we propose our algorithm and derive conditions for obtaining an optimal solution. We prove several important properties used in the design of the proposed algorithm and derive its rate of convergence.

In Section 5, we report all our experimental results and present a detailed illustrative example to show the complete workings of our proposed algorithm for the ease of understanding. We shall also present simulation study to quantify the performance in terms of rate of convergence and solution quality. We highlight our contributions and discuss on possible future extensions in Section 6.

2. Mathematical model

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

2.1. Description of system, jobs, and resources

We consider a generic distributed/parallel computer system shown in Fig. 1. The system consists of n heterogeneous nodes, which represent host computers, interconnected by a generally configured communication/intercommunication network. Each node consists of one or more resources (such as CPU, I/O devices, etc.), contended for by the jobs processed at that node. 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 job can be processed at any node without interruption. Here, we use N to denote the set of nodes, i.e. $n = |N|$ and E to denote a set whose elements are unordered pairs of distinct elements of N. Each unordered pair $e = \langle i,j\rangle$ in E 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 L 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 $V_j = \{i|(i,j) \in E\}$ denote a set of neighboring nodes of node j.

We assume that there are m classes of jobs that can arrive to the system for processing. We denote J as the set of job

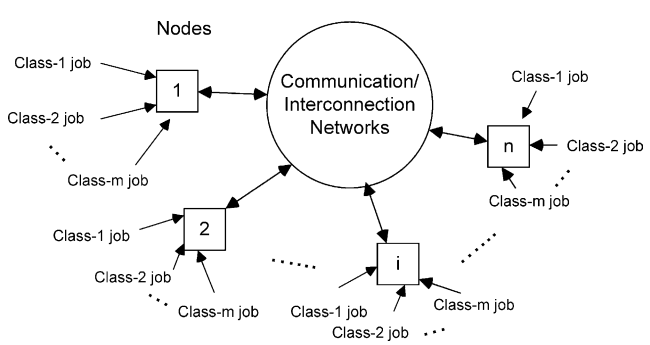

Fig. 1. A distributed/parallel computer system.

classes and let us suppose that each class of jobs has its own priority. For convenience, we define class-1 has the highest priority, class-2 has the second highest, and, so on. In our model, we consider the non-preemptive priority rule whereby a job undergoing process in the node is allowed to complete process without interruption even if a job of higher priority arrives in the mean time. When the node becomes free, the first job of the highest priority is considered for processing. Further, we assume that Class-k jobs arrive at node i, $i \in N$ according to an ergodic process, such as Poisson process, with the average external job arrival rate of ϕ_i^k. A class-k job arriving at node i may either be processed locally or transferred through the network to another node for remote processing. We denote β_i^k as the rate at which class-k jobs are processed at node i, which is also referred to as the rate at which class-k jobs are *processed* at node i.

In general, computing on a network based environment incurs additional overheads such as communication delays due to the underlying links. In a generic treatment of a load balancing problem, it is imperative to account such additional overheads that influence the overall performance. In our model, we assume that there is a communication delay incurred when a class-k job is transferred from one node to the other in the system. Further, we assume that each link (i,j) can transfer the load at its own transmission capability (otherwise referred to as transmission rate, commonly expressed as bytes/s). We denote C as the set of transmission capabilities of all the links and c_{ij} as the transmission capability of a link (i,j), $c_{ij} \in C$. In Ref. [16], there are many delay models proposed for data networks. Without loss of generality, we choose M/M/1 model for the network. Let x_{ij}^k be the class-k job flow-rate from node i to node j, and $x_{ij} = \sum_{k=1}^{m} x_{ij}^k$, $m = |J|$, which is called the *total traffic of class-k jobs* on link (i,j).

Note that in Li and Kameda's model of multi-class jobs distributed/parallel computer system [3], the class-k jobs transferred from the neighboring nodes and external arriving class-k jobs are treated differently. In their model, for class-k jobs and for each processor, the neighboring processors are divided into four different sets such that, processors in each set send the class-k jobs to a processor i based on certain rules. A brief explanation on this algorithm is given in Appendix A. However, in our model, we relax this assumption and consider all the jobs of class-k that arrive at node i as a cumulative amount of class-k type, regardless of their origin. Our model for a node is as shown in Fig. 2. From this figure, the following conservation equations hold

$$\gamma_i^k = \phi_i^k + \sum_{j\in V_i} x_{ji}^k = \beta_i^k + \sum_{j\in V_i} x_{ij}^k, \tag{1}$$

where γ_i^k is the total class-k jobs arriving rate at node i. Thus, we can say that in order to obtain load balancing, each node i must determine β_i^k and each $x_{ij}^k (j \in V_i)$, according to γ_i^k, satisfying Eq. (1). The MRT of a job in the system is also influenced by the mean nodal delay (due to queuing

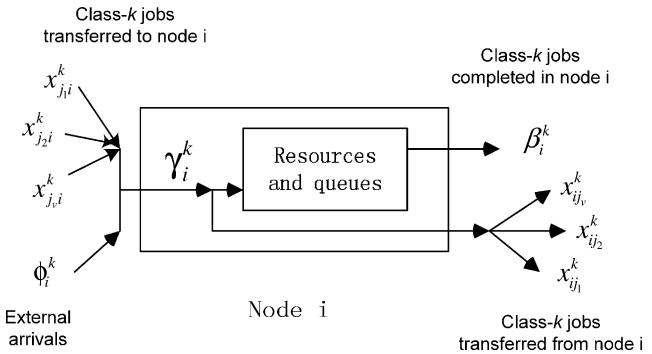

Fig. 2. Job flow in node i.

and processing delays) at the processing node in addition to a (possible) mean communication delay incurred during job transfer phase. Hence, the load balancing policy should determine the values of β and x, where $\beta = [\beta_1, \ldots, \beta_i, \ldots, \beta_n]$, $x = [x_{ij}]$ with $\beta_i = [\beta_i^1, \ldots, \beta_i^m]$, and $x_{ij} = [x_{ij}^1, \ldots, x_{ij}^m]$, $m = |J|$.

Further, we denote $P_i^k(\beta_i)$ as the mean node delay for a class-k job at node i and $G_{ij}^k(x)$ as the mean communication delay for a job transfer from node i to j. Some generic models of $G_{ij}^k(x)$ include the delay of sending a class-k job from node i to node j and the delay of sending the response back from node j to node i. In general, the path taken in each of the above-mentioned transfers (job and response transfers) may be different. Note that the profile of functions $Pik(\beta i)$ and $G_{ij}^k(x)$ may be very complicated. In practice, for analytical ease, it is often assumed that the functions $P_i^k(\beta_i)$ and $G_{ij}^k(x)$ are differentiable, increasing and convex functions [8,16,20].

Let $D(\beta, x)$ denote the MRT of jobs averaged over all classes, which is the mean time a job spends in our system from the time of its arrival. That is MRT, the mean time difference between the time instant at which a job arrives to the system and the time instant at which it departs the system after processing. As in Ref. [5], we obtain

$$D(\beta, x) = \frac{1}{\Phi} \sum_{k \in J} \left\{ \sum_{i \in N} \beta_i^k P_i^k(\beta_i^k) + \sum_{(i,j) \in L} x_{ij}^k G_{ij}^k(x_{ij}^k) \right\} \qquad (2)$$

where $\Phi = \sum_{i \in N} \sum_{k \in J} \phi_i^k$.

Our objective is to balance the loads that arrive to our system such that the MRT (performance measure) is minimized. Note that the MRT is influenced by β_i^k and the transfer rates x_{ij}^k, in order to minimize the MRT. A formal definition of the problem addressed in this paper is presented in Section 3. Load balancing is done by transferring loads from heavily loaded nodes to lightly loaded nodes. However, in doing so a number of factors, such as job arrival rates, processing capabilities of a node, transmission capabilities of a link, etc. must be taken into account. Also, one must avoid a situation wherein a load simply traverses across several nodes without getting processed. Other performance measures of interest, such as a weighted sum of MRTs relative to jobs entering at different nodes, can also be considered in a similar fashion. A simple example of a three-node system is shown in Fig. 3.

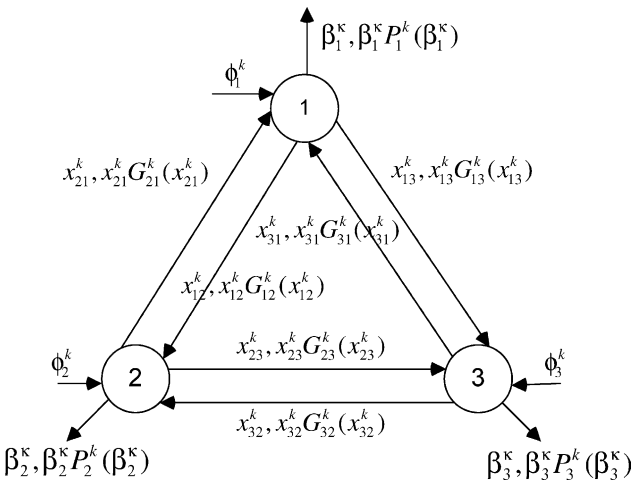

Fig. 3. Example of job flow and its delay.

It may be noted that, although we only use MRT as the main objective in this paper, one can consider other metrics to quantify the performance in our problem formulation. For example, if we consider a min–max metric besides MRT, we can keep track of the number of times a job has been transferred. Thus, if it exceeds a predefined maximum, the job will not be transferred any more. Hence, we can limit the number of transfers of the jobs within some scale. It is beyond the scope of this paper to consider this metric.

3. Problem formulation and solution approach

In this section, based on the model introduced in Section 2, 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 MRT defined in Eq. (2). We state the following

$$\text{Minimize } D(\beta, x) = \frac{1}{\Phi} \sum_{k \in J} \left\{ \sum_{i \in N} \beta_i^k P_i^k(\beta_i^k) + \sum_{(i,j) \in L} x_{ij}^k G_{ij}^k(x_{ij}^k) \right\} \qquad (3)$$

subject to:

$$\phi_i^k + \sum_{j \in V_i} x_{ji}^k = \beta_i^k + \sum_{j \in V_i} x_{ij}^k, \quad k \in J, \quad i \in N,$$

$$\phi_i^k \geq 0, \quad k \in J, \quad i \in N, \qquad x_{ij}^k \geq 0, \quad (i,j) \in L, \quad k \in J.$$

Thus, the solution to our problem lies in determining the optimal values of β and x, respectively. In all the earlier studies [3,8,12,20], the solution to Eq. (3) is obtained by using the method of Lagrange multipliers and key idea is to determine the set of Lagrange multipliers to obtain an optimal solution. In order to find the Lagrange multipliers, *linear section* searches were used, such as *Golden Section*

Search [12], which needs fairly very long computational time. Below we shall present our solution approach.

3.1. Proposed solution

In this section, we propose our solution approach to the problem defined in Eq. (3). As a first step, we add a *virtual* node, which is referred to as the *destination* node (node d), into our network system. Also, we connect node d with each node i, $i \in N$, by a direct link (i,d). Note the virtual node d and each direct link (i,d) do not exist in the real system. The job flow in our 'new' system is shown in Fig. 4. According to this modification, we redefine the set of links L in the system as $L = \{(i,j), |i,j \in N\} \cup \{(i,d), |i \in N\}$ and denote x_{id}^k as the class-k job flowing on link (i,d), which is equal to β_i^k. After these modifications, we introduce another function F_{ij}^k to unite the two different delay functions of P_i^k, the mean node delay of class-k jobs, and G_{ij}^k, the mean communication delay of class-k jobs. Function F_{ij}^k is expressed as the mean link delay of class-k job on link (i,j) as follows

$$F_{ij}^k(x_{ij}^k) = \begin{cases} P_i^k(x_{ij}^k), & (i,j) \in L \text{ and } j = d \\ G_{ij}^k(x_{ij}^k), & (i,j) \in L \text{ and } j \neq d \end{cases}$$

where d is the virtual destination node.

Then, Eq. (2) can be rewritten as:

$$\begin{aligned} D(\beta, x) &= \frac{1}{\Phi} \sum_{k \in J} \left\{ \sum_{i \in N} \beta_i^k P_i^k(\beta_i^k) + \sum_{(i,j) \in L, j \neq d} x_{ij}^k G_{ij}^k(x_{ij}^k) \right\} \\ &= \frac{1}{\Phi} \sum_{k \in J} \left\{ \sum_{i \in N} x_{id}^k F_{id}^k(x_{id}^k) + \sum_{(i,j) \in L, j \neq d} x_{ij}^k F_{ij}^k(x_{ij}^k) \right\} \\ &= \frac{1}{\Phi} \sum_{k \in J} \left\{ \sum_{(i,j) \in L} x_{ij}^k F_{ij}^k(x_{ij}^k) \right\} \end{aligned} \quad (4)$$

Thus, from Eq. (4), we can describe the process of load balancing in another way. As shown in Fig. 4, the three-node system has been transformed into a datagram network, in which node i, $i \in N$, basically acts as a *router*. The way in which the loads are shared by the nodes can be described as follows. Class-k jobs that arrive at node i according to a Poisson process with an average external job arrival rate of ϕ_i^k are routed to destination node d via every node $j \neq i$. This can be understood from Fig. 4. Referring to this figure, we observe that for each node i, $i = 1, 2, 3$, there exist five paths to reach node d. For example, from node 1 to node d, the paths are: $1 \to 2 \to d$, $1 \to 3 \to d$, $1 \to 2 \to 3 \to d$, $1 \to 3 \to 2 \to d$, $1 \to d$, respectively. Thus, node i must determine a set of paths independently for the class-k jobs arriving at it via every other node to node d. Also, note that the class-k jobs may spend some time in the system due to the link delays through the path from node i to node d. In practice, it is reasonable to assume that the mean link delay $F_{ij}^k(x)$ of link (i,j) depends only on the job flow rates x_{ij} on link (i,j), where $x_{ij} = [x_{ij}^1, \ldots, x_{ij}^m]$, $m = |J|$. Thus, our goal of load balancing can be alternatively (and equivalently) stated as a problem which attempts to minimize the mean link delay for each job in the system and our method basically *transforms* the load balancing problem into a routing problem with all the jobs in the system having the same destination of node d. At the same time, we treat the job as a single entity traveling on the links. Notice that if function $D(x)$ is strictly convex, we have the unique solution to problem (4). Since we do not assume that $D(x)$ is strictly convex, the solution may not be unique. However, we can get an optimal solution that minimizes the MRT $D(x)$ although it may not be a unique solution.

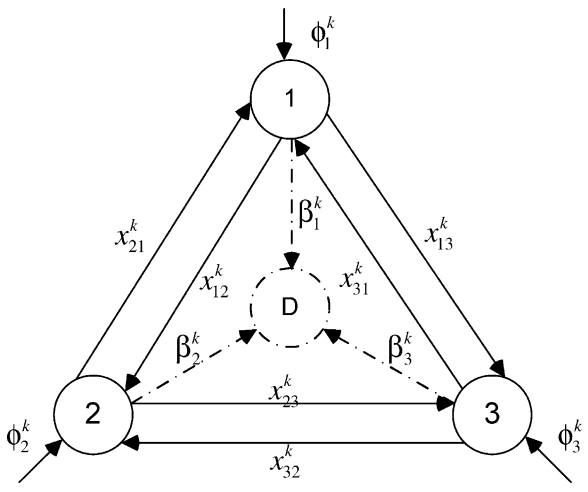

Fig. 4. Job flow in a system with a virtual node.

4. Proposed algorithm and an optimal solution

Once the load balancing problem has been transformed into a routing problem, there are several algorithms proposed in the literature to solve such problems [16–19]. However, in our problem context, it is important to establish the correspondence between the problems of routing and load balancing. Since our solution methodology proposes to use routing to balance the load in the network, we refer to our algorithm as load balancing via virtual routing (LBVR). We shall first present this below.

4.1. Correspondence between routing and load balancing

Below, we identify certain key equivalences between these two problems in a systematic fashion.

Firstly, routing decisions can be made depending on whether the network uses datagrams or virtual circuits. In a datagram network, two successive packets of the same S–D pair (source and destination nodes) may travel along different routes, and a routing decision is necessary for each individual packet. In a virtual circuit network, a routing decision is made at the time of setting up a virtual circuit. The routing algorithm is used to choose the communication

path for the virtual circuit. All packets of the virtual circuit subsequently use this path up to the time that the virtual circuit is terminated. There are two main performance measures that are substantially affected by a routing algorithm—*throughput* and *average packet delay*. In the routing context, the through-put is simply the amount of traffic from a source node to the destination node. In the load balancing problem, throughput is equivalent to amount of processed load by the system and an average packet delay is equivalent to the MRT of jobs in the system. Thus, in our transformed problem defined by Eq. (4), a routing decision may aid to balance the loads that traverse via different nodes before reaching the destination d. Hence, a set of nodes in each path generated by a node i to reach d will be considered as potential receivers in our load balancing problem.

Secondly, another classification of routing algorithm relates to whether there is any route change in response to the traffic input patterns. In *static* routing algorithms [23], the path used by the sessions of each S–D pair is fixed regardless of traffic conditions. Further, in *adaptive* or *dynamic* routing [26,27], the paths used to route new traffic between S–D pairs change occasionally in response to network congestion. For load balancing, static algorithms do not depend on the current state of the nodes in the system, and dynamic policies offer the possibility of improving load distribution at the expense of additional communication and processing overheads [23,33].

In our problem context, by adding a virtual destination node d into our system, we combine the problems of routing and load balancing together as explained by the above correspondence. Now, we can clearly take advantage of the routing algorithms for load balancing.

4.2. Optimal solution

Here we propose our algorithm by using some concepts underlying a routing problem.

When a class-k job arrives at node i, it has many routing paths to choose to reach the destination node d (as can be seen from Fig. 4). We can observe that if the number of nodes increases, the potential routing paths for node 1 will increase dramatically. However, when we consider the decentralized policy each node i in the system only knows about the flow status of link $(i,j), j \in V_i$, its own load x_{id}^k and the loads of its neighboring nodes x_{jd}^k, respectively. Hence, the number of routing paths for node i can be reduced to $(v+1)$ where $v = |V_j|$. This means that a job arriving at a node i can reach node d directly through link (i,d), or it can go to node d via node j, $j \in V_i$ through link (i,j) and link (j,d). An example of paths for each node in a three-node system is shown in Fig. 5, where $p(i)_l$ means the lth path of node i. Because of the distributed working style, each node in the system determines the routing paths according to its neighboring nodes and calculates its transfer rates and processing rate by itself. When node j receives some jobs from node i, node j may retransfer some jobs to another

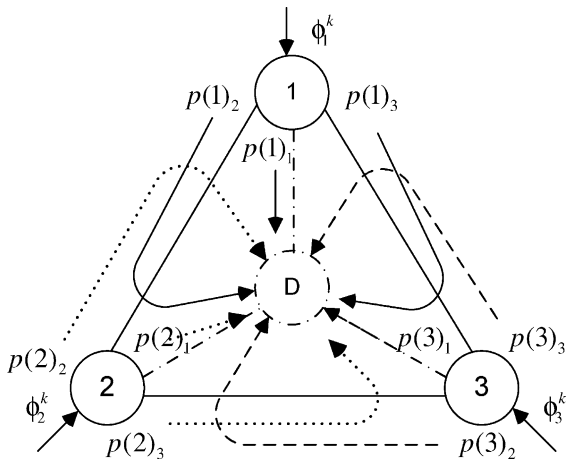

Fig. 5. Routing paths for each node.

node, and so on. Eventually, the jobs come from node i may be processed in one node which is unknown to node i. However, it may be noted that this does not mean that the search of eligible paths is restricted to the neighboring nodes, since some other routing paths with two or more hops can also be added according to the situation.

We denote P_i^k as the set of paths from node i to destination node d for class-k job and hence, we obtain $|P_i^k| = v+1$, $v = |V_i|$. Further, let x_i^k be class-k job path flow vector of node i, where $x_i^k = \{x_p^k, |p \in P_i^k, i \in N\}$. Here, x_p^k is the job flow of path p for class-k job.

We define $D_{ij}^k(x_{ij}^k) = x_{ij}^k F_{ij}^k(x_{ij}^k)$ and refer to this as a *delay function*. Substituting this delay function in Eq. (4), we obtain

$$D(x) = \frac{1}{\Phi} \sum_{k \in J} \left\{ \sum_{(i,j) \in L} D_{ij}^k(x_{ij}^k) \right\} \tag{5}$$

where x_{ij}^k is the total amount of class-k job flow carried by link (i,j) and is given by:

$$x_{ij}^k = \sum_{\text{all paths } p \text{ containing } (i,j)} x_p^k \tag{6}$$

From Eq. (5), it is easy to observe that the function D is an increasing function. Now, we can completely rewrite the optimization problem as follows:

Minimize $D(x)$

$$= \frac{1}{\Phi} \sum_{k \in J} \left\{ \sum_{(i,j) \in L} D_{ij}^k \left(\sum_{\text{all paths } p \text{ containing } (i,j)} x_p^k \right) \right\} \tag{7}$$

subject to:

$$\sum_{p \in P_i^k} x_p^k = \gamma_i^k, \quad i \in N, \quad k \in J \tag{8}$$

$$\gamma_i^k = \phi_i^k + \sum_{j \in V_i} x_{ji}^k, \quad i \in N$$

$$x_p^k \geq 0, \quad p \in P_i^k, \quad i \in N$$

In the following, we will characterize the proposed algorithm in terms of the first derivatives $(D_{ij}^k)'$ of the functions D_{ij}^k. We denote $D(x)$ as the MRT function of Eq. (7) and $\partial D(x)/\partial x_p^k$ as the partial derivative of D with respect to x_p^k. Then

$$\frac{\partial D(x)}{\partial x_p^k} = \frac{1}{\Phi}\sum_{k\in J}\sum_{\text{all links } (i,j) \text{ on path } p}(D_{ij}^k)' = \frac{1}{\Phi}\left\{\sum_{\text{all links } (i,j) \text{ on path } p}\frac{\partial D_{ij}^1(x_{ij}^1)}{\partial x_{ij}^k} + \cdots + \sum_{\text{all links } (i,j) \text{ on path } p}\frac{\partial D_{ij}^m(x_{ij}^m)}{\partial x_{ij}^k}\right\} \quad (9)$$

where the first derivatives $(D_{ij}^k)'$ are evaluated at a value equal to the total amount of class-k job flows corresponding to x^k and $m = |J|$. It is seen that $\partial D/\partial x_p^k$ is the length of path p when the length of each link (i,j) is taken to be the first derivative $(D_{ij}^k)'$ evaluated at x_p^k. Consequently, in what follows $\partial D/\partial x_p^k$ is called the *first derivative length of path p*.

Theorem 1. *The set of values of $\bar{x}$ is an optimal solution to problem (7) only if job flow travels along minimum first derivative length (MFDL) paths for each S–D pair of class-k job, $k \in J$. In addition, if D_{ij}^k are assumed to be convex, then $\bar{x}$ is optimal if and only if job flow travels along MFDL for each S–D pair of class-k jobs, $k \in J$.*

Proof. We shall first prove the first part of the above theorem. Let $\bar{x} = \{\bar{x}^1, \ldots, \bar{x}^k\}$, $k \in J$, be an optimal path flow vector. Then, for class-k job flow, if $\bar{x}_p^k > 0$ for some path p of an S–D pair P_i^k, we must be able to transfer a small amount $\delta > 0$ from path p to any other path p' of the same S–D pair without decreasing the MRT. Otherwise, the optimality of $\bar{x}$ would be violated. For the first derivative, the change in MRT from this transfer is

$$\delta\frac{\partial D(\bar{x})}{\partial x_{p'}^k} - \delta\frac{\partial D(\bar{x})}{\partial x_p^k}$$

and since this change must be non-negative, we obtain:

$$\bar{x}_p^k > 0 \Rightarrow \frac{\partial D(\bar{x})}{\partial x_{p'}^k} \geq \frac{\partial D(\bar{x})}{\partial x_p^k}, \quad \text{for all } p' \in P_i^k \quad (10)$$

In other words, optimal path flow is positive only on paths with a MFDL. Furthermore, at an optimum, the paths along which the input flow γ_i^k of a S–D pair (i,d), $i \in N$, that are different must have equal lengths (and less than or equal to the length of all other paths of S–D pair (i,d)).

Thus, the condition (10) is a *necessary* condition for optimality of $\bar{x}$. The above proof is also valid for the *only if* part of the second part of the theorem for convex functions. Now, the above proof can be back tracked to prove the second part of the theorem when we assume that the functions D_{ij}^k are convex. For example, when the second derivatives $(D_{ij}^k)''$ exist and are positive in the domain of definition of D_{ij}^k. □

Having shown the optimality of the solution, we will now show that the flow on path p is indeed identical to the flow on a link (i,j), when $(i,j) \in p$. This intermediate result will be useful for us later.

Lemma 1. *The class-k job flow on link (i,j), $j \in V_i$, is equal to the class-k job flow on path p, $p \in P_i^k$ and $(i,j) \in p$.*

Proof. Recall the definition of path at the beginning of this section and refer to Fig. 5. We can observe that there is only one path passing through link (i,j), $j \in V_i$, which is $(i \to j \to d)$. From Eq. (6):

$$x_{ij}^k = \sum_{\text{all paths } p \text{ containing } (i,j)} x_p^k$$

$$\Rightarrow x_{ij}^k = x_p^k, \quad (i,j) \in p \text{ and } p \in P_i^k$$

Hence the proof. □

Lemma 2. *In an optimal solution to problem (7), one or both of the class-k job flows on link (i,j) and (j,i) are zero. That is, $x_{ij}^k x_{ji}^k = 0$, where $i \in N$ and $j \in V_i$.*

Proof. This can be proved by contradiction. Assume there exists class-k link traffic in an optimal solution, such as $x_{ij}^k > 0$ and $x_{ji}^k > 0$, where $i,j \in N$ and $j \in V_i$. From Lemma 1, we have:

$$x_{ij}^k = x_p^k > 0, \quad (i,j) \in p \text{ and } p \in P_i^k$$

$$x_{ji}^k = x_{p'}^k > 0, \quad (j,i) \in p' \text{ and } p' \in P_j^k$$

From Theorem 1, we can deduce that path p is a MFDL path for a S–D pair (i,d) and path p' is the MFDL path for S–D pair (j,d), where p is the path given by $(i \to j \to d)$ and p' is the path given by $(j \to i \to d)$.

Consider node i first. Let p_i be the path (i,d) and $p_i \in P_i^k$. Since path p is a MFDL path in P_i^k and $\bar{x}$ is an optimal solution, we obtain:

$$\frac{\partial D(\bar{x})}{\partial x_{p_i}} \geq \frac{\partial D(\bar{x})}{\partial x_p}$$

$$\Rightarrow \frac{\partial D(\bar{x})}{\partial x_{id}^k} \geq \frac{\partial D(\bar{x})}{\partial x_{ij}^k} + \frac{\partial D(\bar{x})}{\partial x_{jd}^k} \quad (11)$$

Similarly, for node j we obtain

$$\frac{\partial D(\bar{x})}{\partial x_{p_j}} \geq \frac{\partial D(\bar{x})}{\partial x_{p'}}$$

$$\Rightarrow \frac{\partial D(\bar{x})}{\partial x_{jd}^k} \geq \frac{\partial D(\bar{x})}{\partial x_{ji}^k} + \frac{\partial D(\bar{x})}{\partial x_{id}^k} \quad (12)$$

where p_j is the path (j,d) and path $(j,d) \in P_j^k$. Since function D is an increasing function, then $\partial D(\bar{x})/\partial x_{ij}^k > 0$

and $\partial D(\bar{x})/\partial x_{ji}^k > 0$. From Eqs. (11) and (12), we have

$$\frac{\partial D(\bar{x})}{\partial x_{id}^k} > \frac{\partial D(\bar{x})}{\partial x_{jd}^k} \text{ and } \frac{\partial D(\bar{x})}{\partial x_{jd}^k} > \frac{\partial D(\bar{x})}{\partial x_{id}^k}$$

which is a clear contradiction and hence, the lemma. □

Lemma 3. *An optimal solution to problem (7) is cycle free, i.e. there exists no class-k link traffic such that* $x_{i_1 i_2}^k > 0$, $x_{i_2 i_3}^k > 0, \ldots, x_{i_m i_1}^k > 0$, *where* $i_1, i_2, \ldots, i_m \in N$.

Proof. We again prove this by contradiction. Assume that there is a class-k link traffic in an optimal solution, such as $x_{i_1 i_2}^k > 0$, $x_{i_2 i_3}^k > 0, \ldots, x_{i_m i_1}^k > 0$, where $i_1, i_2, \ldots, i_m$ are indices of distinct nodes. Considering $x_{i_1 i_2}^k$ and $x_{i_2 i_3}^k$, since function D is an increasing function and according to Theorem 1 and Lemma 1, for links (i_1, d) and (i_2, d), we obtain:

$$\frac{\partial D(\bar{x})}{\partial x_{i_1 d}^k} > \frac{\partial D(\bar{x})}{\partial x_{i_2 d}^k}$$

Recursively applying the proof technique used in Lemma 2 using Eq. (11), we obtain

$$\frac{\partial D(\bar{x})}{\partial x_{i_1 d}^k} > \frac{\partial D(\bar{x})}{\partial x_{i_2 d}^k}$$

$$\vdots$$

$$\frac{\partial D(\bar{x})}{\partial x_{i_m d}^k} > \frac{\partial D(\bar{x})}{\partial x_{i_1 d}^k}$$

which is a clear contradiction and hence the lemma. □

Lemmas 2 and 3 signify the fact that, in an optimal solution, the class-k job flow-rate from the node i to node j cannot be transferred back to node i directly from node j or through other nodes. This is indeed an important result: it is guaranteed that no job that entered the system shall leave without getting processed.

4.3. Design of the proposed algorithm

In Section 4.2, it was shown that an optimal solution results only if job flow travels along MFDL paths for each S–D pair. Equivalently, a set of path flows is strictly suboptimal only if there is a positive amount of flow that travels on a non-MFDL path. This suggests that a suboptimal solution can be improved by transferring some flow to an MFDL path from other paths for each S–D path. However, by transferring all the flow of each S–D pair to the MFDL path will lead to an oscillatory behavior. Thus, it is more appropriate to transfer only part of the flow from other paths to the MFDL path. Below, we shall determine the amount of these flows from each of the other paths, between an S–D pair, to be transferred to MFDL to seek an optimal solution. To do this, we propose our algorithm based on *Newton's method* [14], which is

$$x^{(r+1)} = x^{(r)} - \alpha^{(r)} \left[\frac{\partial^2 D(x^{(r)})}{(\partial x_i)^2} \right]^{-1} \frac{\partial D(x^{(r)})}{\partial x_i}, \tag{13}$$

$r = 0, 1, \ldots$ and $i = 1, \ldots, n$

where $\alpha^{(r)}$ is a positive scalar step-size at iteration r, determined according to some rule [20].

Consider the optimization problem (7). Assume that the second derivatives of D_{ij}^k, denoted by $D_{ij}^{k\,\prime\prime}(x_{ij}^k)$, are positive for all x_{ij}^k. Let $(x^k)^{(r)} = \{(x_p^k)^{(r)}\}$ be the class-k job path flow vector obtained after r iterations, and let $\{(x_{ij}^k)^{(r)}\}$ be the corresponding set of total amount of class-k job link flows. For each S–D pair (i, d), let $\bar{p}_i^k$ be an MFDL path for class-k job, $k \in J$.

The optimization problem (7) can be converted (for the purpose of the next iteration) to a problem involving only active (positive) constraints by expressing the flows of MFDL paths $\bar{p}_i^k$ in terms of the other path flows, while eliminating the passive (equality) constraints (8)

$$\sum_{p \in P_i^k} x_p^k = \gamma_i^k, \qquad i \in N, \ k \in J$$

in the process. For each S–D pair (i, d), $x_{\bar{p}_i^k}^k$ is substituted in the cost function $D(x^k)$ using the equation

$$x_{\bar{p}_i^k}^k = \gamma_i^k - \sum_{\substack{p \in P_i^k \\ p \neq \bar{p}_i^k}} x_p^k \tag{14}$$

thereby obtaining a redefined problem of the form

$$\begin{aligned} &\text{Minimum} && \tilde{D}(\tilde{x}) \\ &\text{subject to } x_p^k \geq 0, && \text{for all } p \in P_i^k,\ p \neq \bar{p}_i^k,\ i \in N \end{aligned} \tag{15}$$

where $\tilde{x}$ is the vector of all class-k job path flows which are not MFDL paths.

We calculate the derivatives that will be needed to apply the iteration (13) to the problem defined by Eq. (15). Using Eq. (14) and the definition of $\tilde{D}(\tilde{x})$, we get

$$\frac{\partial \tilde{D}(\tilde{x}^{k(r)})}{\partial x_p^k} = \frac{\partial D(x^{k(r)})}{\partial x_p^k} - \frac{\partial D(x^{k(r)})}{\partial x_{\bar{p}_i}^k} \tag{16}$$

for all $p \in P_i^k$, $p \neq \bar{p}_i^k$, $i \in N$, where $x^{k(r)}$ is the vector x^k in the rth iteration and $\partial D(x)/\partial x_p$ is the first derivative length of path p, that is given by Eq. (9). Regarding second derivatives, a straightforward differentiation of the first derivative expressions (16) and (9) shows that

$$\frac{\partial^2 \tilde{D}(\tilde{x}^{k(r)})}{(\partial x_p^k)^2} = \frac{1}{\Phi} \sum_{l \in J} \sum_{(i,j) \in L_p} D_{ij}^{l\,\prime\prime}(x_{ij}^{k(r)})$$

$$= \frac{1}{\Phi} \left\{ \sum_{(i,j) \in L_p} \frac{\partial^2 D_{ij}^1(x_{ij}^1)}{(\partial x_{ij}^k)^2} + \cdots + \sum_{(i,j) \in L_p} \frac{\partial^2 D_{ij}^m(x_{ij}^m)}{(\partial x_{ij}^k)^2} \right\}$$

$m = |J|$, and for all $p \in P_i^k$, $p \neq \bar{p}_i^k$, $i \in N$, where, for each p, L_p is a set of links belonging to either p or to the corresponding MFDL path $\bar{p}_i^k$, but not in both. This can be realized from Eq. (16).

Table 1
Proposed load balancing algorithm (LBVR)

Step 1: *Initialization*

$r = 0$. (r: iteration index)
For $k = 1$ to m, $m = |J|$
  For $i = 1$ to n, $n = |N|$
    Determine P_i^k, the set of class-k job flow paths of node i.
    Calculate the rate of class-k jobs at node i, that is, $\gamma_i^{k(0)} = \phi_i^k + \sum_{j \in V_i} x_{ji}^{k(0)}$,
    where $x_{ji}^{k(0)} = 0$, $j \in V_i$.
    Find a feasible $(x_p^{k(0)})$ as an initial feasible solution, which satisfies $\sum_{p \in P_i^k} x_p^{k(0)} = \gamma_i^{k(0)}$. Here, we choose $x_{p_i^k}^{k(0)} = \gamma_i^{k(0)}$ and $x_p^{k(0)} = 0$, where $p \in P_i^k$ *and* $p \neq p_i^k$, p_i^k is the path $(i \to d)$ for class-k job.
  End For
End For

Step 2: *Solution Procedure*

Let $r = r + 1$
For $k = 1$ to m, $m = |J|$ do the following for class-k jobs.
  For $i = 1$ to n, $n = |N|$ do the following for node i.
    Let $x^{k(r)} = x^{k(r-1)}$.
    Calculate the class-k jobs at node i, i.e., determine $\gamma_i^{k(r)}$
    For $j = 1$ to $v + 1$, $v = |V_i|$
      Calculate the first derivative lengths of the paths p, that is $d_p^{(r)}$, $p \in P_i^k$
    End For
    Find the minimum first derivative length (MFDL) paths $\bar{p}_i^{k(r)}$ among P_i^k.
    For $j = 1$ to v, $v = |V_i|$
      To path p_j, $p_j \in P_i^k$, $p_j \neq \bar{p}_i^{k(r)}$, calculate the second derivative length: $H_{p_j} = \frac{1}{\Phi} \sum_{l \in J} \sum_{(i,j) \in L_p} D_{ij}^{l}{}''(x_{ij}^{k(r)})$.
      Let $x_{p_j}^{k(r)} = max\{0,\ x_{p_j}^{k(r-1)} - \alpha^{(r-1)} H_{p_j}^{-1}(d_{p_j}^{(r-1)} - d_{\bar{p}_i^{k(r-1)}}^{(r-1)})\}$
      Let $x_{ij}^{k(r)} = x_{p_j}^{k(r)}$, *link* $(i,j) \in p_j$
    End For
    Let $x_{\bar{p}_i^{k(r)}}^{k(r)} = \gamma_i^{k(r)} - \sum_{\substack{p \in P_i^k \\ p \neq p_i^{k(r)}}} x_p^{k(r)}$ and $x_{ij}^{k(r)} = x_{\bar{p}_i^{k(r)}}^{k(r)}$, *link* $(i,j) \in \bar{p}_i^{k(r)}$
  End For
End For

Step 3: *Stopping rule*

If $|D(x^{(r)}) - D(x^{(r-1)})|/D(x^{(r)}) < \varepsilon$, then **stop**, where ε is a desired acceptable tolerance for solution quality; otherwise, go to Step 2.

Expressions for both the first and second derivatives of the 'reduced' cost $\tilde{D}(\tilde{x})$, are available and thus the scaled projection method given by Eq. (13) can be applied. The iteration takes the form

$$x_p^{k(r+1)} = \max\{0, (x_p^{k(r)} - \alpha^r H_p^{-1}(d_p - d_{\bar{p}_i^k}))\},$$
$$\text{for } p \in P_i^k, \;\; p \neq \bar{p}_i^k, \;\; i \in N \tag{17}$$

where d_p and $d_{\bar{p}_i^k}$ are the first derivative lengths of the paths p and $\bar{p}_i^k$ and can be rewritten using Eq. (9) as

$$d_p = \frac{1}{\Phi} \sum_{l \in J} \sum_{\text{all links } (i,j) \text{ on path } p} D_{ij}^{l\prime}(x_{ij}^{k(r)}),$$

$$d_{\bar{p}_i^k} = \frac{1}{\Phi} \sum_{l \in J} \sum_{\text{all links } (i,j) \text{ on path } \bar{p}_i^k} D_{ij}^{l\prime}(x_{ij}^{k(r)})$$

and H_p is the 'second derivative length' given by

$$H_p = \frac{1}{\Phi} \sum_{l \in J} \sum_{(i,j) \in L_p} D_{ij}^{l\prime\prime}(x_{ij}^{k(r)})$$

used in Eq. (17). The step-size α^r is some positive scalar which may be chosen by a variety of methods. From our rigorous simulation test runs, we choose α^r to be a constant from the range [0.5,1] for all r, which is shown to work well in our method.

The proposed algorithm LBVR transforms the basic load balancing problem into a routing problem and it is based on Newton's method (13). Notice that the class-k jobs at node i, $\gamma_i^{k(r)}$ may vary in every iteration. We choose Newton's method, because using this method, an optimal solution can be implemented naturally when the input rates are time varying, as discussed in Ref. [16]. Further, Newton's method becomes an obvious choice for our problem context owing to the style adopted in our problem formulation. As it is important to investigate on the rate of convergence for such an optimization problem, we will discuss the complexity of convergence of our algorithm in Section 4.4. Due to the varying input conditions ($\gamma_i^{k(r)}$, in every iteration), it may be possible that in some iteration, $x_{\bar{p}_i^k}^{k(r)} = \gamma_i^{k(r)} - \sum x_p^{k(r)} < 0$, where $p \in P_i^k$ and $p \neq \bar{p}_i^{k(r)}$. If this happens, then this will violate the constraint $\bar{x}_{p_i^k}^k \geq 0$ and may be detrimental in the next iteration. To circumvent this problem, we iterate *Step 2* without changing the values of k and i, till the constraint $\bar{x}_{p_i^k}^k \geq 0$ is satisfied and then carry out the next iteration.

Our algorithm has some interesting and important features. The first one is due to its simple and feasible implementable design structure. Though the paths of each S–D pair (i, d) are recomputed before the main calculation phase, we can add or delete some paths according to the current status of the system during the course of main iteration. The algorithm can be implemented in a decentralized fashion in the sense that each node can use the algorithm independently and depends only on local information collected from neighboring nodes. The second important feature is that it can be applied in a *dynamic* load balancing environment with little modification. Finally, the computational time of our algorithm is significantly small. Note the complexity of the algorithm for each node is $O(v + 1) \approx O(v)$ ($v = |V_i|$). To implement this algorithm, we only need to compute the first and second derivative functions of $D(x)$, which can be easily obtained. We will determine the exact computational time of our algorithm in Section 5. A pseudo-code of LBVR algorithm is shown in Table 1.

4.4. Rate of convergence

We denote $h(x^r) = D'(x^r)$. If an optimal solution is at point $\bar{x}$, then we have $h(\bar{x}) = 0$. Alternatively, we have:

$$\phi(x^r) = x^r - \alpha^r \frac{h(x^r)}{h'(x^k)}$$

By the mean value theorem, we obtain

$$x^{r+1} - \bar{x} = \phi(x^r) - \phi(\bar{x}) = \phi'(\xi^r)(x^r - \bar{x}),$$

where ξ^r lies between x^r and $\bar{x}$. Then

$$|x^{r+1} - \bar{x}| = \frac{|h''(\xi^r)h(\xi^r)|}{[h'(\xi^k)]^2} |x^r - \bar{x}|$$

and

$$|h(\xi^r)| = |h(\xi^r) - h(\bar{x})| = |h'(\eta^r)||\xi^r - \bar{x}| \leq |h'(\eta^r)||x^r - \bar{x}|$$

where η^r lies between ξ^r and $\bar{x}$. Hence:

$$|x^{r+1} - \bar{x}| \leq \frac{|h''(\xi^r)h|}{[h'(\xi^r)]^2} |x^r - \bar{x}|^2$$

Let

$$\beta = \sup \frac{|h''(\xi^r)h'(\eta^r)|}{[h'(\xi^r)]^2},$$

then

$$|x^{r+1} - \bar{x}| \leq \beta |x^r - \bar{x}|^2$$

Hence the convergence of our algorithm is 2, which exhibits a *super-linear* convergence. The workings of the proposed algorithm is shown in Fig. 6.

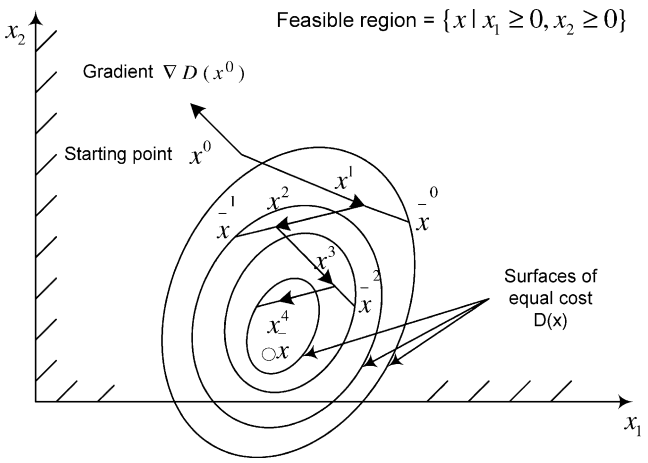

Fig. 6. Operation of the proposed algorithm.

5. Experimental results and discussions

Our algorithm LBVR retains two important features of practical interest: one is its decentralized style of working and the other is its simple structure, in terms of implementation ease. To appreciate one of the strengths of our algorithm, consider Fig. 3. In this figure, if the heavily loaded node 1 wants to send some jobs to lightly loaded node 3, when the failure of link (1,3) is occurred. In this case, our proposed algorithm LBVR considers adding a multi-hop routing path $(1 \rightarrow 2 \rightarrow 3)$ for node 1 to take advantage of additional information on link (2,3). Hence, LBVR algorithm is more robust in its working style (implementation). As a matter of fact, with the approach proposed in Ref. [3] node 1 still cannot send jobs to node 3 *directly* via node 2 due to the link failure. At the same time, based on Golden Section Search, LK algorithm obtains a local optimal solution in each iteration and it is proven that by carrying such iterations, the global optimal solution can be eventually obtained. However, in Refs. [16,20], 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.

The work reported in Ref. [3] proposes an algorithm (LK) and compares with well-known standard steepest descent and FD algorithms. It was proven that algorithm LK performs much better than the well-known FD algorithm [21] in terms of its computational time (faster convergence). Thus, it is wiser to compare the performance with LK algorithm in terms of algorithmic convergence rate and basic design. For the purpose of continuity and as a quick reference, we have described the LK algorithm, in brief, in Appendix A. Secondly, we use our proposed algorithm LBVR to study an optimal multi-class job load balancing problem in a distributed system, with a detailed illustrative example showing the complete workings. We present a detailed discussions on the performance of our algorithm.

5.1. Demonstration of LBVR algorithm: example of load balancing

We shall now show an illustrative example which demonstrates the details of the workings of our LBVR algorithm.

Consider a distributed computer system with nine host computers as shown in Fig. 7. In such a system, we assume that there are two classes of jobs ($k = 1, 2$) which arrive at node i according to a time-invariant Poisson process. The inter-arrival times of jobs of each class are mutually independent. A job of class-1 is always served prior to a job of class-2. Within each class the queue discipline is first-come-first-served. A class-1 job arriving when no other jobs of its class are present will wait until the job (if any) of class-2 being served completes its service. The arrival processes of all classes are assumed independent,

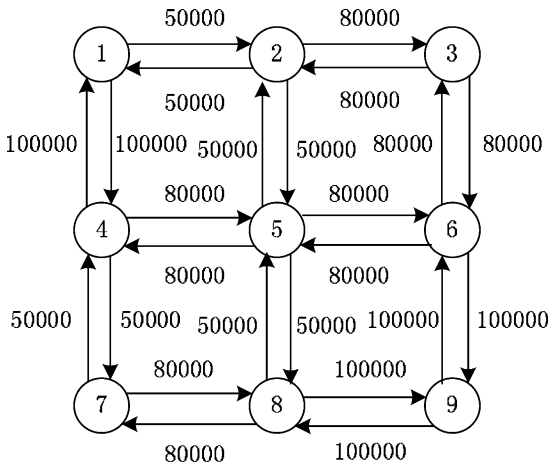

Fig. 7. An example of nine-nodes distributed computer system.

Poisson, and independent of service times. We assume node i is an M/G/1 system with one CPU and one queue. The first two moments of service time of each class-k at node i are denoted $\bar{X}_i^k = 1/\mu_i^k$ and $\bar{X}_i^{k2} = 2/(\mu_i^k)^2$, respectively, where μ_i^k is the average service time of class-k job of node i.

For node i, we will derive an equation for average delay for each priority class, which is similar to the $P-K$ formula [13]. We denote N_i^k as the average number of jobs in queue for priority k, W_i^k as the average queuing time for priority k, $\rho_i^k = \beta_i^k/\mu_i^k$ as the system utilization for priority k, and R as mean residual service time, respectively. We assume that the overall system utilization is less than 1, that is

$$\rho_i^1 + \rho_i^2 + \cdots + \rho_i^m < 1$$

When this assumption is not satisfied, there will be some priority class-k such that the average delay of customers of priority k and lower will be infinite while the average delay of customers of priority higher than k will be finite.

As in the derivation of the $P-K$ formula, we have, for the highest priority class:

$$W_i^1 = R + \frac{1}{\mu_i^1} N_i^1$$

Eliminating N_i^1 from this equation using *Little's Theorem* [16], $N_i^1 = \beta_i^1 W_i^1$, we obtain

$$W_i^1 = R + \rho_i^1 W_i^1$$

and finally

$$W_i^1 = \frac{R}{1 - \rho_i^1}$$

For class-2 job, we have a similar expression for the queuing delay W_i^2 except that we have to count the additional queuing delay due to jobs of class-1 that arrive while a job is waiting in queue. This is the meaning of the last term in the above formula.

$$W_i^2 = R + \frac{1}{\mu_i^1} N_i^1 + \frac{1}{\mu_i^2} N_i^2 + \frac{1}{\mu_i^1} \beta_i^1 W_i^2$$

Using Little's Theorem ($N_i^k = \beta_i^k W_i^k$), we finally obtain:

$$W_i^2 = \frac{R}{(1-\rho_i^1)(1-\rho_i^1-\rho_i^2)}$$

The mean residual service time R can be derived from $P-K$ formula as:

$$R = \frac{1}{2}\sum_{k=1}^{m} \beta_i^k \bar{X}_i^{k2}$$

The average delay per job of class-k in node i is then given by:

$$T_i^k = \frac{1}{\mu_i^k} + W_i^k$$

Finally, we obtain

$$T_i^1 = \frac{1}{\mu_i^1} + \frac{\sum_{k=1}^{2} \beta_i^k \bar{X}_i^{k2}}{2(1-\rho_i^1)} \tag{18}$$

$$T_i^2 = \frac{1}{\mu_i^2} + \frac{\sum_{k=1}^{2} \beta_i^k \bar{X}_i^{k2}}{2(1-\rho_i^1)(1-\rho_i^1-\rho_i^2)} \tag{19}$$

as the mean node delays per job of class-1 and class-2, respectively, in node i.

Table 2 gives the values of the parameters of the node models we used in this numerical example. Using Eqs. (18) and (19) and the values of the parameters, we obtain the functions of total delay for the class-1 and 2 job rates. For example, for node 1

$$\beta_1^1 F_1^1(\beta_1^1) = D_{1d}^1(x_{1d}^1) = x_{1d}^1\left(\frac{1}{6} + \frac{x_{1d}^1/18 + x_{1d}^2/18}{2(1-x_{1d}^1/6)}\right)$$

$$\beta_1^2 F_1^2(\beta_1^2) = D_{1d}^2(x_{1d}^2) = x_{1d}^2\left(\frac{1}{6} + \frac{x_{1d}^1/18 + x_{1d}^2/18}{2(1-x_{1d}^1/6)(1-x_{1d}^1/6 - x_{1d}^2/6)}\right)$$

where $1 - x_{1d}^1/6 - x_{1d}^2/6 > 0$.

For the communication links, we use the M/M/1 model. For the sake of simplicity, we assume that there is no difference between class-1 and class-2 jobs as far as link model is concerned. Hence, the job flow-rate for link (i,j), $j \neq d$ is $x_{ij} = x_{ij}^1 + x_{ij}^2$. Assume that the transmission capacity of the link (i,j) is c_{ij} (packets/s). Assume that, on an average, Q_1 packets are required to describe a job and that, on average, Q_2 packets are required in sending back a response for a job. Here, we choose $Q_1 = Q_2 = Q = 100$. Hence:

$$x_{ij}G_{ij}(x_{ij}) = \frac{Qx_{ij}}{c_{ij} - Qx_{ij} - Qx_{ji}} + \frac{Qx_{ij}}{c_{ji} - Qx_{ji} - Qx_{ij}} \tag{20}$$

The first part of Eq. (20) signifies the communication delay of job flow on link (i,j) and the second part signifies the communication delay of the responses sent back from node j to i. The capacities of each link in packets-per-second are shown in Fig. 7. The average external job arrival rate ϕ_i^k for class-1 and class-2 job are shown in Table 3.

By adding a virtual node d into the system and connecting node i and node d with link (i,d), we obtain Fig. 8. Using the node delay model (18), the communication delay model (20), and the load of two classes of jobs for each node shown in Table 3, we obtain the experimental results for LK algorithm and our algorithm LBVR, as shown in Fig. 9. Table 4 shows the processing rates at node i.

From Table 4, we observe that both the LK and LBVR algorithms deliver more-or-less identical performance in terms of balancing the load (processing rate). However, Fig. 9 elicits a significant advantage of LBVR over LK in terms of rate of convergence. From this figure, we can observe that to obtain the similar MRT of each job (as exemplified in Table 3), the computational time of LK algorithm is more

Table 2
Parameter values of node model

Node (i)	μ_i^1 (jobs/s)	$\bar{X}_i^{12} = 2/(\mu_i^1)^2$	μ_i^2 (jobs/s)	$\bar{X}_i^{22} = 2/(\mu_i^2)^2$
1	6	1/18	6	1/18
2	5	2/25	5	2/25
3	7	2/49	5	2/25
4	6	1/18	4	1/8
5	5	2/25	6	1/18
6	5	2/25	4	1/8
7	6	1/18	5	2/25
8	5	2/25	4	1/8
9	5	2/25	6	1/18

Table 3
The average external job arrival rate of class-1 and class-2 jobs

Node (i)	ϕ_i^1 (packets/min)	ϕ_i^2 (packets/min)
1	40	60
2	50	40
3	20	30
4	50	80
5	70	60
6	60	70
7	10	50
8	60	65
9	30	40

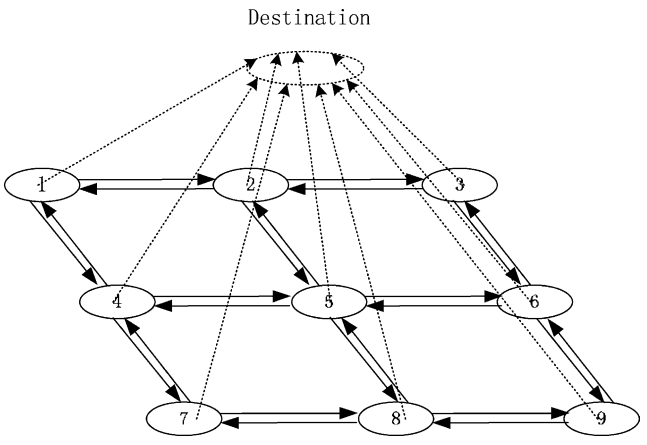

Fig. 8. The distributed computer system with a virtual node d.

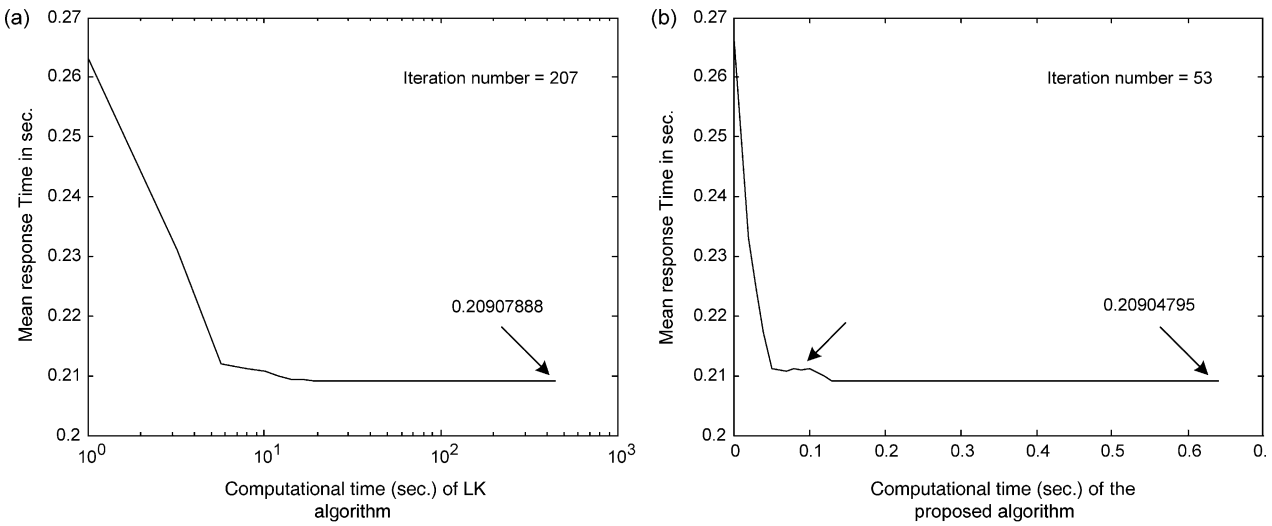

Fig. 9. Comparing the algorithms in computational time.

than 400 s, whereas LBVR consumes less than 1 s. This clearly shows that the proposed algorithm has much faster convergence rate. This significant improvement in performance mainly comes due to the computational nature of LBVR algorithm. In LK algorithm, much of the time (close to 90%) is spent in Golden-section search phase and the remaining time is spent in determining certain inverse functions by means of recursive computations (as these do not have closed-form solutions).

Further, a close scrutiny on the behavior of LBVR from Fig. 9, identified by an *arrow A*, reveals the following information. It may be observed that close to this interval, there is a little fluctuation in the MRT. The reason for this fluctuation is that, in some cases, the LBVR algorithm goes out of feasible solution space due to the choice of step-size α. The algorithm, then has to take some time (consuming certain iterations) to re-enter the feasible solution space and to become normal. As the step-size choice affects the computational time, based on several rigorous test runs, we fixed the value of α to be within the [0.5,1] range for better accurate results.

6. Conclusions

A novel super-linear static load balancing algorithm, referred to as LBVR, using the concept of data *routing* in computer networks, has been proposed in this paper. The objective is to minimize the MRT of the jobs that arrive to a distributed system for processing. Our formulation considers a multi-class jobs that arrive at processors of a distributed network system for processing. The jobs are considered for processing in a non-preemptive manner. In this paper, we have proposed a novel solution to the load balancing problem defined by Eq. (3). Several constraints such as, job flow-rate, communication delays due to the underlying network, processing delays at the processors, are considered in the problem formulation. We formulated the problem of load balancing as a constrained non-linear minimization problem, and as a solution approach, we have transformed the problem into an equivalent problem of routing under the above set of constraints. The solution approach that was proposed is conclusively proven to yield an optimal solution, when the delay function defined in Section 4.2 is assumed to be a convex function. Whereas, when the delay function was a non-convex function, a *necessary* condition was derived to obtain an optimal solution to the problem. Further, the design of the algorithm involves a systematic application of routing technique, through which a set of potential receivers, of an unbalanced quantum of load, is identified. This routing technique is shown to deliver a super-linear rate of convergence. The rate of convergence to optimal solution by our algorithm when compared with Ref. [3] was experimented and the results are shown to testify the theoretical findings. Finally, we have also shown that our algorithm guarantees that no job is left unprocessed once it enters the system for processing and the solution approach is cycle-free (Lemma 3).

Although the studies attempted in Refs. [3,5] are the ones that are considered close to our modeling, there exist some

Table 4
Completed job rate (β_i^k) in node i

Node (i)	LK algorithm		LBVR algorithm	
	Class-1 (jobs/s)	Class-2 (jobs/s)	Class-1 (jobs/s)	Class-2 (jobs/s)
1	0.000000	2.064288	0.000000	2.063941
2	0.000000	1.262917	0.000000	1.219627
3	2.463605	0.000000	2.480405	0.000000
4	1.695861	0.000000	1.683883	0.000000
5	0.000000	2.108337	0.000000	2.062730
6	0.815314	0.000000	0.837796	0.000000
7	0.684354	0.833333	0.672499	0.833333
8	0.840866	0.000000	0.825417	0.000000
9	0.000000	1.981124	0.000000	2.070368

key differences. In Ref. [5], while the formulation was generic, only one class of jobs is considered. In Ref. [3] multi-class of jobs are considered under a more-or-less similar formulation. However, in the latter case, the underlying node model was different, in the sense that, each node i treats its neighboring nodes differently according to the transfer vector of class-k jobs. Whereas, in our model, we relax this constraint and handle a generic situation in which the entire set of class-k jobs arriving at node i (from other nodes and also externally at node i), are considered as a whole while balancing the entire load.

The current problem context poses the following challenges to be considered as possible future extensions. First attempt can be in tuning the formulation to handle the dynamic case of load balancing, wherein network routes and established paths may dynamically change. Since the concept of routing is fundamentally used in our problem formulation, the above-mentioned proposed extension seem to be possible to attempt. Secondly, additional constraints such as limited resources availability, such as buffer capacity at a node, bandwidth availability of channels, etc. can be considered in the problem formulation. Even in this case, owing to our faster solution convergence, these can be attempted. Lastly, problem formulation and solution for handling jobs with time critical constraints (jobs having deadlines) can also be considered.

Acknowledgements

The authors would like to thank Professor Jie Li, University of Tsukuba, Japan, for his valuable suggestions to their work in terms of clarifying certain key issues and implementation aspects of the algorithm reported in Ref. [3]. The comparison study conducted in this paper used the LK algorithm and the authors thank his guidance in carrying out an exact implementation of their algorithm. The authors would also like to thank the anonymous reviewers for their valuable suggestions, which improved readability and presentation.

Appendix A. LK algorithm in brief

To obtain an optimal solution to problem (3) with its constraints, LK algorithm basically forms the Lagrangian functions as follows

$$H(\beta, x, \alpha) = \Phi D(\beta, x) + \sum_{k\in J}\sum_{i\in N}\alpha_i^k\left(\phi_i^k + \sum_{j\in V_i} x_{ji}^k - \sum_{j\in V_i} x_{ij}^k - \beta_i^k\right)$$

where α_i^k ($i \in N$, $k \in J$) are the Lagrange multipliers. The solution to the problem is to determine the values of α_i^k. The set of neighboring nodes of node i (V_i) are divided into the following four sets for each class-k in the optimal solution:

(1) Class-k active source nodes (A_i^k). Node j in this set sends a part of class-k arriving jobs to node i and also processes the other part of class-k jobs locally.
(2) Class-k idle source nodes (Id_i^k). Node j in this set sends all class-k arriving jobs to node i.
(3) Class-k neutral nodes (Nu_i^k). Node j in this set does not send (or receive) any class-k jobs to (or from) node i.
(4) Class-k sink nodes (S_i^k). Node j in this set receives some class-k jobs from node i.

In LK Algorithm, in each iteration, the algorithm uses the Golden section search (e.g. Ref. [12]) to find the values of α_i^k. Once α_i^k are determined, the values of β_j^k, x_{ji}^k, x_{ij}^k and β_i^j, $j \in V_i$ can be directly obtained by the equations deduced in Ref. [3]. The stopping rule is similar with our algorithm.

In addition, the authors compare their algorithm with the well-know FD algorithm. The FD algorithm is a standard steepest-descent algorithm, which evaluates from a feasible solution in the steepest descent direction and chooses a step-size that minimizes the mean job flow time along such a direction to form a new feasible solution in each iteration. Then, they show that their algorithm has much faster convergence to the minimum response time than FD algorithm. We omit further details.

References

[1] N.G. Shivaratri, P. Krueger, M. Singhal, Load distributing for locally distributed systems, Computer 25 (12) (1992) 33–44.
[2] L. Anand, D. Ghose, V. Mani, ELISA: an estimated load information scheduling algorithm for distributed computing system, Computers and Mathematics with Applications 37 (1999) 57–85.
[3] J. Li, H. Kameda, Load balancing problems for multiclass jobs in distributed/parallel computer systems, IEEE Transactions on Computers 47 (3) (1998) 322–332.
[4] J. Watts, S. Taylor, A practical approach to dynamic load balancing, IEEE Transactions on Parallel and Distributed Systems 9 (3) (1998) 235–248.
[5] K.W. Ross, D.D. Yao, Optimal load balancing and scheduling in a distributed computer system, Journal of the Association for Computing Machinery 38 (3) (1991) 676–690.
[6] Y. Amir, B. Awerbuch, A. Barak, R. Sean Borgstrom, A. Keren, An opportunity cost approach for job assignment in a scalable computing cluster, IEEE Transactions on Parallel and Distributed Systems 11 (7) (2000) 760–768.
[7] G. Manimaran, C. Siva Ram Murthy, An efficient dynamic scheduling algorithm for multiprocessor real-time systems, IEEE Transactions on Parallel and Distributed Systems 9 (3) (1998) 312–319.
[8] A.N. Tantawi, D. Towsley, Optimal static load balancing in distributed computer systems, Journal of the Association for Computing Machinery 32 (2) (1985) 445–465.
[9] B. Shirazi, A.R. Hurson, K. Kavi, Scheduling and Load Balancing in Parallel and Distributed Systems, IEEE Computer Society Press, Los Alamitos, CA, 1995.
[10] A.E. Kostin, I. Aybay, G. Oz, A randomized contention-based load-balancing protocol for a distributed multiserver queuing system, IEEE

Transactions on Parallel and Distributed Systems 11 (12) (2000) 1252–1272.
[11] M. Mitzenmacher, The power of two choices in randomized load balancing, IEEE Transactions on Parallel and Distributed Systems 12 (10) (2001) 1094–1104.
[12] D.J. Wilde, C.S. Beightler, Foundations of Optimization, Prentice-Hall, Englewood Cliffs, NJ, 1967.
[13] N.U. Prabhu, Foundations of Queuing Theory, Kluwer Academic Publishers, Dordrecht, 1997.
[14] M. Avriel, Nonlinear Programming Analysis and Methods, Prentice-Hall, Englewood Cliffs, NJ, 1997.
[15] M. Mitzenmacher, How useful is old information?, IEEE Transactions on Parallel and Distributed Systems 11 (1) (2000) 6–20.
[16] D. Bertsekas, R. Gallager, Data Networks, Prentice-Hall, Englewood Cliffs, NJ, 1992.
[17] S. Dafermos, The traffic assignment problem for multiclass-user transportation networks, Transportation Science 6 (1972) 73–87.
[18] E. Altman, H. Kameda, Equilibria for multiclass routing in muli-agent networks, Proceeding of the 40th IEEE Conference on Decision and Control, Orlando, USA, December, 2001.
[19] R.K. Ahuja, T.L. Mananti, J.B. Orlin, Network Flows, Prentice-Hall, Englewood Cliffs, NJ, 1993.
[20] D.P. Bertsekas, Nonlinear Programming, Athena Scientific, Belmont, MA, 1995.
[21] L. Fratta, M. Gerla, L. Kleinrock, The flow deviation network design, Networks 3 (1973) 97–133.
[22] Y. Zhang, K. Hakozaki, H. Kameda, K. Shimizu, A performance comparison of adaptive and static load balancing in heterogeneous distributed systems, Proceedings of IEEE 28th Annual Simulation Symposium, Phoenix, Ariz, April, 1995, pp. 332–340.
[23] P.D. Spraggis, D. Towsley, C.G. Cassandras, Optimality of static routing policies in queuing systems with blocking, Decision and Control, 1991., Proceedings of the 30th IEEE Conference, vol. 1, 1991, pp. 809–814.
[24] M.M. Eshaghian (Eds.), Heterogenous Computing, Artech House, Boston: Artech House, 1996.
[25] C. Cunha, P. Kacsuk, S.C. Winter, Parallel Program Development for Cluster Computing: Methodology, Tools and Integrated Environments, Nova Science Publishers, Huntington, NY, 2001.
[26] M.D. Grammatikakis, J.S. Jwo, M. Kraetzl, S.H. Wang, Dynamic and static packet routing on symmetric communication networks, GLOBECOM'94. Communications: The Global Bridge., IEEE 3 (1994) 1591–1595.
[27] H. Tokumaru, K. Kinoshita, N. Yamai, K. Murakami, A fast and efficient dynamic routing for inter-agent communications, Info-tech and Info-net, 2001. Proceedings of ICII 2001—Beijing, vol. 4, 2001, pp. 73–78.
[28] M.J. Zaki, W. Li, S. Parthasarathy, Customized dynamic load balancing for a network of workstations, Journal of Parallel and Distributed Computing 43 (1997) 156–162.
[29] H. Kameda, J. Li, C. Kim, Y. Zhang, Optimal Load Balancing in Distributed Computer Systems, Springer, London, 1996.
[30] J. Li, H. Kameda, A decomposition algorithm for optimal static load balancing in tree hierarchy network configurations, IEEE Transactions on Parallel and Distributed Systems 5 (5) (1994) 540–548.
[31] C. Kim, H. Kameda, An algorithm for optimal static load balancing in distributed computer systems, IEEE Transactions on Computers 41 (3) (1992) 381–384.
[32] J. Li, H. Kameda, Optimal static load balancing of multi-class jobs in a distributed computer system, Proceedings of 10th International Conference on Distributed Computing Systems, 1990, 1990, pp. 562–569.
[33] R. Jain, The Art of Computer System Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling, Wiley, New York, 1991.