On Load Balancing in Multicomputer/Distributed Systems Equipped with Circuit or Cut-Through Switching Capability

Ching-Chih Han, *Member*, *IEEE Computer Society*, Kang G. Shin, *Fellow*, *IEEE*, and Sang Kyun Yun, *Member*, *IEEE Computer Society*

Abstract—For multicomputer or distributed systems that use circuit switching, wormhole routing, or virtual cut-through (the last two are collectively called the *cut-through switching*), the communication overhead and the message delivery time depend largely upon link contention rather than upon the distance between the source and the destination. That is, a larger communication overhead or a longer delivery delay occurs to a message when it traverses a route with heavier traffic than the one with a longer distance and lesser traffic. This characteristic greatly affects the selection of routes for interprocessor communication and/or load balancing. We consider the load-balancing problem in these types of systems. Our objective is to find the maximum load imbalance that can be eliminated without violating the (*traffic*) *capacity constraint* and the route to eliminate the imbalance while keeping the maximum link traffic as low as possible. We investigate the load-balancing problem under various conditions. First, we consider the case in which the excess load on each overloaded node is divisible. We devise a network flow algorithm to solve this type of load balancing problem *optimally* in polynomial time. Next, we impose the realistic assumption that the system uses a specific routing scheme so that the excess load transferred from an overloaded node to an underloaded node must use the route found by the routing scheme. For this case, we use a graph transformation technique to transform the system graph to another graph to which the same network flow algorithm can be applied to solve the load balancing problem optimally. Finally, we consider the case in which the excess load on each overloaded node is indivisible, i.e., the excess load must be transferred as an entity. We show that the load-balancing problem of this type becomes NP-complete and propose a heuristic algorithm as a solution.

Index Terms—Load balancing, minimax flow problem, excess/deficit load, overloaded/underloaded nodes, link traffic.

1 Introduction

In distributed/multicomputer systems, the storage of each computer/node may have been overloaded or underloaded as data and/or files are created and deleted. This is more likely to occur as large files for such applications as multimedia are frequently created/deleted and transferred. Since there is usually limited storage space at a node, uneven data/file distribution may result in inefficient use of storage and affect the ability of future data/file creation. For example, some nodes may not have sufficient space to store new data/files even if the overall system has sufficient space for all the data/files. Load balancing in this respect is thus to transfer the excess (data) load on overloaded nodes to underloaded ones to balance the (data) load among all the nodes in the system. For this load balancing, we need to efficiently transfer the excess load without affecting communications within and/among the existing applications.

For distributed/multicomputer systems that use circuit switching, wormhole routing [1], or virtual cut-through [2], the communication overhead and the message delivery time depend largely upon link contention rather than upon the distance between the message's source and destination. That is, a larger communication overhead or a longer delivery delay results when a message traverses a route with heavier traffic than the one with a longer distance and less traffic. System performance depends largely upon how evenly the traffic is distributed in wormhole routing [3]. This characteristic greatly affects the selection of routes for interprocessor communication (IPC) or load balancing. The objective of selecting a route for IPC or load balancing is thus to balance the network load among all links and reduce the probability of link contention.

The major difference between IPC and load balancing is that, in the former, we must select a route or routes for each pair of communicating processors, while, in the latter, we can select a route or routes from an overloaded node to one or more underloaded nodes. Note that the excess load on an overloaded node can be transferred to *any* underloaded node or nodes, instead of a particular one. Because of this difference, most, if not all, of the variations of the IPC routing problem are NP-hard, while optimal algorithms of polynomial-time complexity exist for several variations of the load-balancing problem. Bianchini and Shen [4] presented a traffic scheduling algorithm that yields optimal network traffic patterns in multiprocessor networks.

- *C.-C. Han is with CreOSys, Inc., 39560 Stevenson Pl., Suite 221, Fremont, CA 94539. E-mail: cchan@creosys.com.*
- *K.G. Shin is with the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109-2122. E-mail: kgshin@eecs.umich.edu.*
- *S.K. Yun is with the Department of Computer Science, Seowon University, Cheongju, Chungbuk, 361-742, Korea. E-mail: skyun@seowon.ac.kr.*

Manuscript received 27 Aug. 1996; revised 12 Aug. 1999; accepted 8 Dec. 1999.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 102090.

Kandlur and Shin [5] studied the route-selection problem for interprocessor communication in multicomputer networks equipped with virtual cut-through switching capability. Bokhari [6] solved the load-balancing problem in circuit-switched multicomputers. He found the largest amount of load imbalance that can be eliminated without contention under several restricted assumptions: 1) There is "unit" load imbalance; 2) there is a fixed routing algorithm; 3) no more than one unit of excess load can be transferred via any link. Moreover, his solution does not take into account the other IPC traffic.

This paper deals with the route-selection problem for balancing the load in multicomputer/distributed systems that use circuit switching, wormhole routing, or virtual cut-through. We assume that load imbalance on each node can be any *arbitrary* value, instead of one unit only, and each link has a capacity constraint. Our main concern is to find the maximum load imbalance that can be eliminated without violating the capacity constraint on each link and to select routes to eliminate the imbalance while keeping maximum link traffic flow as low as possible. While transferring the excess load from overloaded nodes to underloaded ones balances the storage load among all nodes, minimizing the maximum link traffic among all links balances the communication load among all links. Moreover, the remaining link capacities can be used for other communications. Two cases of loads are studied. First, we consider the case in which the excess load on each overloaded node is divisible, i.e., can be arbitrarily divided and transferred to one or more underloaded nodes. Second, we consider the case in which there may be one or more *entities* of excess load on each node and each of them is indivisible and must be transferred to an underloaded node as a single entity. We also take into account the effect of existing IPC traffic on route selection for transferring excess load.

In [6], Bokhari considered multicomputer systems that use some specific routing schemes such as row-column routing in meshes and e-cube routing in hypercubes. He used a graph transformation technique and a network flow algorithm to solve the load-balancing problem in these systems. The graph transformation schemes used for meshes and hypercubes are different. In contrast, we consider multicomputer/distributed systems with and without specific routing schemes and propose a simple, unified graph transformation scheme which transforms a graph with a specific routing scheme into a graph without the specific routing scheme. We solve the load-balancing problem using a network flow algorithm in the graph without the specific routing scheme.

With the proposed graph transformation scheme and the network flow algorithm, we show that, for the case of divisible excess load, the load-balancing problem with or without specific routing schemes can be solved optimally in polynomial time, i.e., we can find the maximum load imbalance that can be eliminated without violating the traffic capacity constraint on each link while minimizing the maximum contention among all links. For the case of indivisible excess load, we first prove that the load-balancing problem is NP-complete and then propose a heuristic algorithm for it.

The rest of the paper is organized as follows: In Section 2, we formally define the load-balancing problem considered in this paper, transform it into a network flow problem, and briefly review a network flow algorithm to solve our load-balancing problem. In Section 3, we solve the load-balancing problem under the assumptions that excess load is divisible and there is no specific routing scheme in the system under consideration. Section 4 shows how to transform the graph representing a system with a specific routing scheme to another graph so that the technique described in Section 3 can be used to find an optimal solution for the load balancing in the system. In Section 5, we give an NP-complete proof and a heuristic algorithm for the load-balancing problem with indivisible excess load. The paper concludes with Section 6.

2 Problem Formulation and a Network Flow Algorithm

2.1 Problem Formulation

The system under consideration is either a distributed point-to-point network or a multicomputer with an interconnection structure, such as a mesh or a hypercube. We will use a directed graph $G = (V, E)$ to represent the system, where the vertex set V represents the set of nodes/processors in the system and the edge set E represents the set of communication links.

When the system needs to perform load balancing, each node is either overloaded, underloaded, or neutral. The excess and deficit loads can be any *arbitrary* values, as opposed to only one unit as assumed in [6]. Let $s_i, 1 \leq i \leq p$, be overloaded nodes and $t_j, 1 \leq j \leq q$ be underloaded nodes. The excess load on an overloaded node $s_i \in V$ is denoted by e_i and the deficit load on an underloaded node $t_j \in V$ is denoted by d_j. As in [6], we assume that the global state of the system and the degree of load imbalance on each node are known to the load balancing controller. We require at most e_i units of load to be transferred from an overloaded node s_i to underloaded nodes and at most d_j units of load to be transferred to an underloaded node t_j from overloaded nodes. We call this requirement the *load transfer constraint*. A traffic capacitor function C is defined on the edge set E, i.e., each link $(v_i, v_j) \in E$ is associated with a traffic capacity $C(v_i, v_j)$, which is the maximum communication volume that can be transferred along the link (v_i, v_j) during load balancing. The *link capacity constraint* restricts the total communication volume on a link (v_i, v_j) from exceeding the link traffic capacity $C(v_i, v_j)$ during load balancing.

The load-balancing problem we consider is to find the maximum load imbalance that can be eliminated without violating any link capacity constraint and to select the routes to eliminate the load imbalance, while keeping maximum link traffic flow as low as possible. During load balancing, minimizing the maximum link traffic balances the communication load among all links.

This load-balancing problem can be transformed into a network flow problem if we construct a new graph G' as

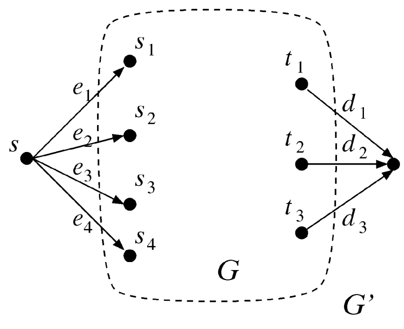


Fig. 1. Graph construction for load balancing.

follows: $G' = (V', E')$, where $V' = V \cup \{s, t\}$, $E' = E \cup \{(s, s_i) \mid s_i$ is an overloaded node, $1 \leq i \leq p\} \cup \{(t_j, t) \mid t_j$ is an underloaded node, $1 \leq j \leq q\}$ as shown in Fig. 1. In other words, a new graph G' is constructed by adding to a graph G a new source node s, a new sink node t, new edges connecting from the source node s to each overloaded node s_i, and new edges connecting from each underloaded node t_j to the sink node t. The traffic capacity $C'(s, s_i)$ on a new edge (s, s_i) is the excess load value e_i, $C'(t_j, t)$ on a new edge (t_j, t) is the deficit load value d_j, and $C'(v_i, v_j)$ for a edge $(v_i, v_j) \in E$ is $C(v_i, v_j)$. The maximum load imbalance that can be eliminated in a graph G is the maximum flow from the source node s to the sink node t in a new graph G' and can be obtained by running a maximum flow algorithm [7]. However, we also want to keep the maximum of link traffic as low as possible and this can be solved not by a maximum flow algorithm, but by the *minimax flow algorithm* [8].

The minimax flow problem is to find a maximum flow for a network that also minimizes the maximum edge cost, where the cost of an edge is defined to be the weight times the flow of the edge. The minimax flow problem with 0/1 weights is a special case of the minimax flow problem in which the weight of each edge is either 0 or 1. The edge cost is the link traffic during load balancing. Since new edges in the new graph G' do not correspond to communication links, their traffic need not be kept as low as possible and their edge cost is zero and, thus, their weights are zero. The cost of an edge corresponding to a communication link is its link traffic and, thus, its weight is one. Therefore, the load-balancing problem in this paper can be transformed into a minimax flow problem with 0/1 weights. Its solution algorithm was proposed in [8].

The system may or may not use a specific routing scheme. The system with a specific routing scheme must be transformed into the system without specific routing schemes as network flow algorithms including a minimax flow algorithm do not consider specific routing schemes.

2.2 The Minimax Flow Algorithm

Before describing our solution for the load-balancing problem, we first give a brief review on the minimax flow problem and algorithm. Details on the network flow problem and the minimax flow problem/algorithm can be found in [9] and [8], respectively. The *minimax transportation problem*, similar to the minimax flow problem, can also be found in [10].

Let $N = (V, E, s, t, C)$ be a network with node set V, edge set E, source s, sink t, and capacity function $C : E \rightarrow R^+ \cup \{0\}$, where $G = (V, E)$ is the underlying directed graph with $|V| = n$ and $|E| = m$, and R^+ is the set of positive real numbers. Each edge $(u, v) \in E$ is also associated with a nonnegative real-valued weight $w(u, v)$. If $w(u, v)$ is either 0 or 1 for all edges $(u, v) \in E$, we say that the network has a 0/1 weight function w.

For convenience, we extend the capacity function and weight function to all vertex pairs by defining $C(u, v) = 0$ and $w(u, v) = 0$ for all $(u, v) \notin E$. A *flow* in a network N is a function $f : V \times V \rightarrow R^+ \cup \{0\}$ that satisfies the following properties:

1. *Capacity constraint*: $0 \leq f(u, v) \leq C(u, v)$, for all $u, v \in V$.
2. *Conservation condition*: $\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v)$, for all $u \in V - \{s, t\}$.

For each edge $(u, v) \in E$, $f(u, v)$ is called the *flow* in (u, v). For each $(u, v) \in V \times V$, $f(u, v) - f(v, u)$ is called the *net flow* from u to v. The capacity constraint states that the flow in (u, v) is bounded by the capacity $C(u, v)$ and the conservation condition states that the net flow going into a node, except the source and the sink, is equal to the net flow going out of the node. The *value* of a flow f, denoted as $|f|$, is the net flow going out of the source, i.e., $\sum_{v \in V} (f(s, v) - f(v, s))$.

If $(u, v) \in E$ and $f(u, v) = C(u, v)$, we say that flow f *saturates* edge (u, v) and call (u, v) an *f-saturated* edge in N. The *cost* (with respect to flow f) of each edge $(u, v) \in E$ is defined to be $w(u, v) \cdot f(u, v)$. The *minimax flow* problem is to find a maximum flow f which minimizes $\max_{(u,v) \in E} w(u, v) \cdot f(u, v)$. Since our load-balancing problem can be transformed to the minimax flow problem with a 0/1 weight function, we shall henceforth concentrate on networks with 0/1 weight functions.

Definition. *Given a network $N = (V, E, s, t, C)$ with a 0/1 weight function w, define $N(\beta) = (V, E^\beta, s, t, C^\beta)$ to be a new network with $E^\beta = E$ and $C^\beta(u, v) = C(u, v)$ if $w(u, v) = 0$ and $C^\beta(u, v) = \min(C(u, v), \beta)$ if $w(u, v) = 1$, for each edge $(u, v) \in E$. An edge $(u, v) \in E^\beta$ is called a* critical *edge if $w(u, v) = 1$ and $C^\beta(u, v) = \beta < C(u, v)$.*

The maximum edge cost allowed for the network $N(\beta)$ is β. Let f^* be a maximum flow in N and f^β a maximum flow in $N(\beta)$. Since $C^\beta(u, v) \leq C(u, v)$ for all $(u, v) \in E$, we have $|f^\beta| \leq |f^*|$ for all $\beta \geq 0$ and, thus, $|f^*|$ is the maximum value of all $|f^\beta|$. Therefore, the minimum value of the maximum edge cost for a maximum flow in N, β^* is the minimum value of β such that $|f^\beta| = |f^*|$.

Definition. $\beta^* = \min\{\beta \mid \beta \geq 0$ *and* $|f^\beta| = |f^*|\}$, *where f^β is a maximum flow in $N(\beta)$.*

We proposed in [8] a minimax flow algorithm, MMC01, as a solution to the minimax flow problem with a 0/1 weight function. MMC01 simply finds β^* and constructs a maximum flow f^{β^*} for the network $N(\beta^*)$. For completeness, we list Algorithm MMC01 in Fig. 2 and summarize it below. However, for the sake of conciseness, we omit the proofs of the correctness and time complexity of the algorithm. The interested reader is referred to [8] for details.

```
Algorithm MMC01

Step 1. Find a maximum flow $f^*$ and its value $|f^*|$ for the network $N = (V, E, s, t, C)$.

Step 2. Let $\ell$ be the number of edges with nonzero weights in $N$ (w.l.o.g. assume $\ell \geq 1$).
        Let $\lambda := \ell + 1$ and $\beta := 0$.

Step 3. Construct network $N(\beta) = (V, E^\beta, s, t, C^\beta)$.
        Find a maximum flow $f^\beta$ and its value $|f^\beta|$ for $N(\beta)$.
        If $|f^\beta| = |f^*|$ go to Step 5.

Step 4. Let $\Delta := |f^*| - |f^\beta|$.
        Let $R$ be the set of $f^\beta$-saturated critical edges in $N(\beta)$, i.e.,
        $R := \{(u, v) \in E^\beta \mid w(u, v) = 1 \text{ and } f^\beta(u, v) = C^\beta(u, v) = \beta < C(u, v)\}$.
        Let $\lambda := \min(|R|, \lambda - 1)$ and $\beta := \beta + \Delta/\lambda$.
        Go to Step 3.

Step 5. A maximum flow, $f^\beta$, that minimizes the maximum edge cost is found, and the maximum
        edge cost with respect to flow $f^\beta$ is $\beta$.
```

Fig. 2. Algorithm for minimax flow problem with a 0/1 weight function.

The idea behind Algorithm MMC01 is that, in each iteration, variable β of the constructed network $N(\beta)$ is set to the maximum edge cost allowed in that iteration. With this maximum edge cost, the capacity of an edge (u, v) with $w(u, v) = 1$ is set to $\min(C(u, v), \beta)$, i.e., the flow allowed to go through edge (u, v) is restricted to $\min(C(u, v), \beta)$. The algorithm repeatedly constructs maximum flows for networks $N(\beta)$ with increasing values of β. Initially, $\beta := 0$ (Step 2). If $|f^0| = |f^*|$, there is a maximum flow with zero cost. Otherwise, if $|f^\beta| = |f^*|$ and $|f^{\beta'}| < |f^*|$ for all $0 \leq \beta' < \beta$, the optimal value of β (i.e., the minimum value of the maximum edge cost, β^*) is found.

In Step 3, if $|f^\beta| < |f^*|$, the optimal value of β has not been found. For each $(u, v) \in E^\beta = E$, if $w(u, v) = 1$ and $f^\beta(u, v) = C^\beta(u, v) = \beta < C(u, v)$, (u, v) is an f^β-saturated critical edge in $N(\beta)$. Therefore, to get a larger flow, we need to increase the capacities of critical edges. Let Δ and λ be defined as in the algorithm (Step 4). It has been shown in [8] that $\beta + \Delta/\lambda \leq \beta^*$. Hence, we set $\beta := \beta + \Delta/\lambda$ and repeat the process. This assignment guarantees that the value of β is always less than or equal to the optimal value β^* and, upon termination, $|f^\beta| = |f^*|$ and, hence, $\beta = \beta^*$.

The time complexity of Algorithm MMC01 is shown in the following theorem.

Theorem 1. *Algorithm MMC01 terminates in at most ℓ iterations and, hence, has a time complexity of $O(\ell \cdot M(n, m))$, where ℓ is the number of edges with nonzero weight and $M(n, m)$ is the time complexity of the algorithm to find a maximum flow in a network with $|V| = n$ vertices and $|E| = m$ edges and $1 \leq \ell \leq m$.*

Proof. See [8]. □

If the capacity C and flow f in N are not nonnegative real numbers but nonnegative integers, Algorithm MMC01 can still be applied to find a (integral) minimax flow for the network N except that the statement $\beta := \beta + \Delta/\lambda$ in Step 4 should be changed to $\beta := \beta + \lceil \Delta/\lambda \rceil$.

3 Systems without Specific Routing Schemes

In this section, we discuss the load-balancing problem for systems without being constrained by any specific routing scheme, i.e., the excess load to be transferred from an overloaded node to an underloaded node can use any path between them.

We first consider the case in which the excess load on each overloaded node can be arbitrarily divided and transferred to one or more underloaded nodes along different routes. In the case where excess load is indivisible, i.e., each overloaded node may have one or more entities of excess load each of which can only be transferred to an underloaded node as an entity, the load-balancing problem becomes more difficult. We will discuss this case in Section 5.

For the case that excess load is arbitrarily divisible, Algorithm MMC01, described in Section 2.2, can be easily applied to find the maximum amount of load imbalance that can be eliminated and to select the routes for load balancing, minimizing the maximum link traffic. An algorithm for load balancing of divisible excess loads consists of two steps. In Step 1, we construct a new graph as described in Section 2.1. In Step 2, we use the MMC01 algorithm described in Fig. 2 to find a minimax flow f. This algorithm is listed in Fig. 3.

```
Algorithm DIV-MMC01

Given a graph $G = (V, E)$, where

Step 1. Construct a network $N = (V', E', s, t, C')$, where $V' = V \cup \{s, t\}$, $E' = E \cup \{(s, s_i) \mid 1 \leq i \leq p\} \cup \{(t_j, t) \mid 1 \leq j \leq q\}$, $C'(u, v) = C(u, v)$ for $(u, v) \in E$, $C'(s, s_i) = e_i$ for $1 \leq i \leq p$, and $C'(t_j, t) = d_j$ for $1 \leq j \leq q$. Let $w(u, v) = 1$ for $(u, v) \in E$, and otherwise, $w(u, v) = 0$.

Step 2. Use Algorithm MMC01 to find a minimax flow $f$ for $N$.
```

Fig. 3. Algorithm for balancing divisible excess loads.

When there is additional communication traffic generated by existing applications, Algorithm DIV-MMC01 should be modified. Let $F(u,v)$ be the estimated communication traffic volume by current applications during load balancing. Since the capacity of link (u,v) that the load-balancing traffic can use is reduced from $C(u,v)$ to $C(u,v) - F(u,v)$, the network $N(\beta)$ in Algorithm MMC01 of Phase II should be redefined as follows:

Definition. *Let $N = (V', E', s, t, C')$, w, and F be defined as above. Define $N(\beta) = (V', E^\beta, s, t, C^\beta)$ to be a new network with $E^\beta = E'$ and $C^\beta(u,v) = C'(u,v) - F(u,v)$ if $w(u,v) = 0$ and $C^\beta(u,v) = \min(C'(u,v) - F(u,v), \beta - F(u,v))$ if $w(u,v) = 1$, for each edge $(u,v) \in E$. An edge $(u,v) \in E^\beta$ is called a* critical *edge if $w(u,v) = 1$ and*

$$C^\beta(u,v) = \min(C'(u,v) - F(u,v), \beta - F(u,v)) < C'(u,v) - F(u,v).$$

Moreover, the initial value of β in Step 2 of MMC01 should be changed to $\max_{(u,v)\in E} F(u,v)$. In this case, the value of β in each iteration of MMC01 is the maximum link traffic allowed for that iteration, where link traffic include both existing communication traffic and load balancing traffic, $F(u,v) + f(u,v)$.

Our algorithm is compared with the algorithm that uses a maximum flow algorithm instead of MMC01 in Step 2 of Algorithm DIV-MMC01. We applied the two algorithms to an 8×8 mesh network. It is assumed that the capacity of every link is 1,000 and there are four overloaded nodes and four underloaded nodes. Every underloaded node has a deficit load of 2,000 and the excess loads of all overloaded nodes have the same size. The size of excess load was varied from 0 to 1,300. The maximum amount of load imbalance that can be eliminated by our algorithm is the same as that by the maximum flow algorithm. However, the maximum link flow in our algorithm is much lower than that in maximum flow algorithm except in cases of large excess loads, as shown in Fig. 4.

4 Systems with Specific Routing Schemes

In this section, we discuss the load-balancing problem for systems with specific routing schemes. In a deterministic routing algorithm or a partially adaptive routing algorithm, selection of the routing path is restricted to a subset of possible paths and selection of an output link can be restricted by the input link. For example, the row-column (or XY) algorithm in meshes routes a packet first along the X dimension and then along the Y dimension. In this algorithm, a packet input along the Y dimension cannot be

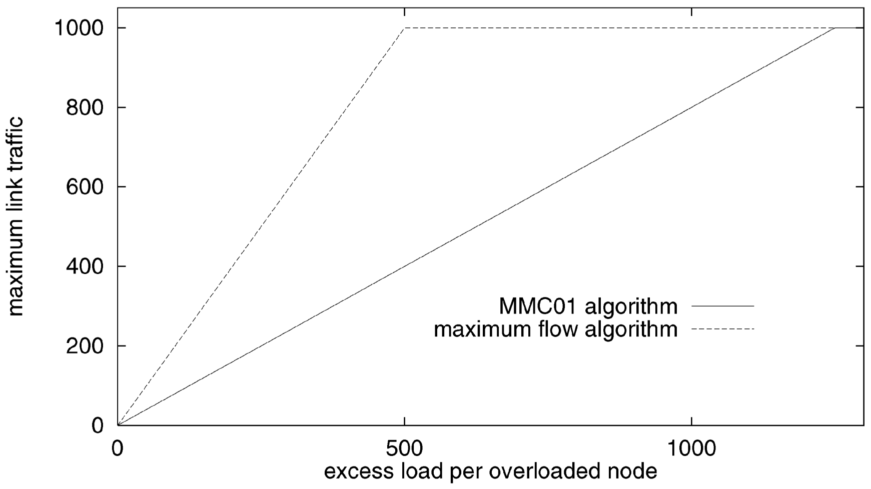


Fig. 4. Maximum link flows when excess load is divisible.

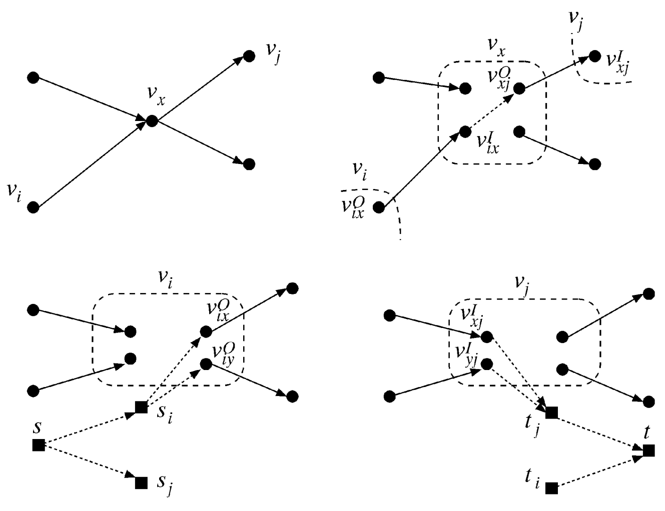


Fig. 5. Illustration of graph transformation.

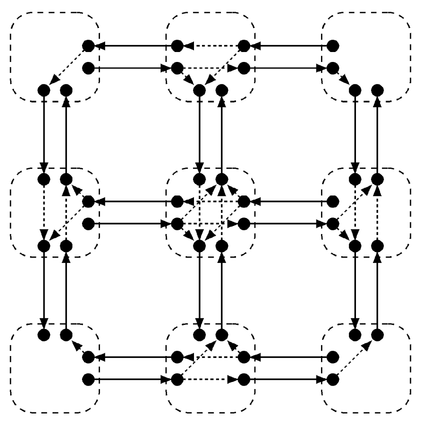


Fig. 6. The transformed graph of a 3×3 mesh that uses the row-column routing scheme.

forwarded along the X dimension. The e-cube routing algorithm in hypercubes routes a packet first along the lowest dimension and then along the higher dimensions, and cannot forward a packet along lower dimension than its input dimension. The west-first routing algorithm in meshes based on the turn model [11] routes a packet first west, if necessary, and then adaptively south, east, north. In the west-first algorithm, a packet input along a dimension other than west cannot be transferred along west. We need to transform a graph with a specific routing scheme into a graph with no specific routing scheme in order to prevent flow from being forwarded along a path that is impossible under a specific routing when a network flow algorithm is executed.

Given a system graph $G = (V, E)$ with a specific routing scheme, we can transform G into another graph $G' = (V', E')$ without a specific routing scheme according to the following rules (see Fig. 5):

R1. Each node $v_x \in V$ is split into $d(v_x)$ nodes in G', where $d(v_x)$ is the total degree of node v_x. If $(v_i, v_x) \in E$, there is a node $v_{ix}^I \in V'$, which is called an input splitting node of v_x, and if $(v_x, v_j) \in E$, there is a node $v_{xj}^O \in V'$, which is called an output splitting node of v_x.

R2. For each edge $(v_i, v_x) \in E$, there is a corresponding edge $(v_{ix}^O, v_{ix}^I) \in E'$ with the capacity $C'(v_{ix}^O, v_{ix}^I) = C(v_i, v_x)$ and the weight of 1. (This edge corresponds to a real communication link.)

R3. If the path along two consecutive edges (v_i, v_x) and (v_x, v_j) is possible, there is an edge $(v_{ix}^I, v_{xj}^O) \in E'$ with the capacity $C(v_{ix}^I, v_{xj}^O) = \infty$ and the weight of 0. We call this edge an internal edge.

R4. There is a source node $s \in V'$. For each overloaded node $v_i \in V$, there are a node $s_i \in V'$ and an edge (s, s_i) with the capacity $C(s, s_i) = e_i$ and the weight of 0, where e_i is the excess load on v_i. For each output splitting node $v_{ix}^O \in V'$ of a node v_i, there is an edge $(s_i, v_{ix}^O) \in E'$ with the capacity $C(s_i, v_{ix}^O) = e_i$ and the weight of 0.

R5. There is a sink node $t \in V'$. For each underloaded node $v_j \in V$, there are a node $t_j \in V'$ and an edge (t_j, t) with the capacity $C(t_j, t) = d_j$ and the weight of 0, where d_j is the deficit load on v_j. For each input splitting node $v_{xj}^I \in V'$ of a node v_j, there is an edge $(v_{xj}^I, t_j) \in E'$ with the capacity $C(v_{xj}^I, t_j) = d_j$ and the weight of 0.

After the system graph G is transformed into G', we can treat the system represented by G' as one without any specific routing scheme and solve the load-balancing problem by finding a minimax flow for the network $N = (V', E', s, t, C')$ with the weight function w as described in Section 3, where $G = (V', E')$ and C' and w are the capacity and weight functions defined in rules **R1**-**R5**.

The transformed graphs (obtained by applying only rules **R1**-**R3**) of a 3×3 mesh that uses the row-column routing scheme are shown in Fig. 6. Each node in three-dimensional hypercubes with the e-cube routing and each node in meshes with the west-first routing are transformed as shown in Fig. 7. Note that we assume each link between two adjacent nodes u and v in a mesh or a hypercube is a bidirectional communication link and, thus, there are two directed edges (u, v) and (v, u) corresponding to this link in the graph representation of the mesh or the hypercube.

5 Systems with Indivisible Excess Loads

In this section, we discuss the case in which the excess load is indivisible. We assume that there is no specific routing scheme in the system. For systems that use certain specific routing schemes, one can first apply the graph transformation rules described in Section 4 to the representing graph

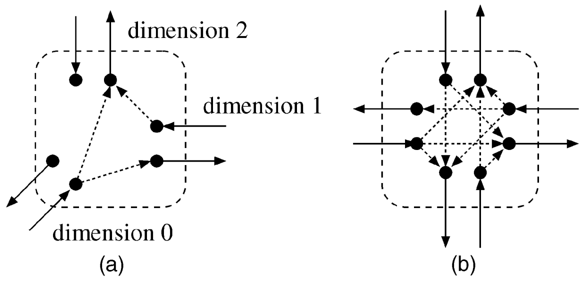


Fig. 7. The transformation of nodes. (a) A node in a 3-cube with e-cube routing. (b) A node is a mesh with west-first routing.

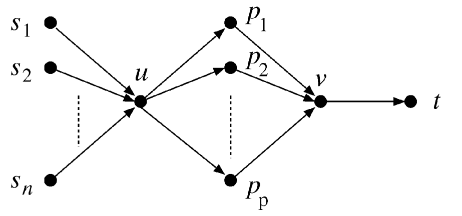


Fig. 8. Instance construction in the NP-completeness proof.

and then treat the transformed graph as a system with no specific routing scheme.

As discussed in Section 3, in a system with indivisible excess load, each overloaded node s_i has one or more indivisible entities of excess load $e_{i1}, e_{i2}, \ldots, e_{ik_i}$, for some $k_i \geq 1$, each of which can only be transferred to an underloaded node as an entity. Without loss of generality, we assume that each overloaded node s_i has exactly one entity of indivisible excess load e_i since if s_i has $k_i > 1$ entities of excess load, we can add a new overloaded node s_{ij} with one entity of indivisible excess load e_{ij} and a new edge (s_{ij}, s_i) for each entity of excess load e_{ij}, $1 \leq j \leq k_i$, of s_i and treat s_i as a neutral node.

We first show that the load-balancing problem with indivisible excess load is NP-hard in the strong sense [12] (in fact, we show that the problem of finding the maximum load imbalance that can be eliminated without considering the link contention is already NP-hard in the strong sense if the excess load is indivisible). We then propose a heuristic algorithm as a solution to the NP-hard case of the load-balancing problem. The decision version of the load-balancing problem of finding the maximum load imbalance that can be eliminated is to ask, given a number B, whether or not it is possible to eliminate at least B units of load imbalance (without violating the link capacity and load transfer constraints).

Theorem 2. *The decision version of the load-balancing problem of finding the maximum load imbalance that can be eliminated is NP-complete in the strong sense if the excess load is indivisible.*

Proof. It is easy to see that the decision version of the load-balancing problem is in NP. To complete the proof, we reduce to it the *multiprocessor scheduling problem* [12]: Given a set $\mathbf{A} = \{a_1, a_2, \ldots, a_n\}$ of n tasks, a length $l(a_i)$ for each $1 \leq i \leq n$, a number p of processors, and a deadline D, is there a partition $\mathbf{A} = A_1 \cup A_2 \cup \cdots \cup A_p$ of $\mathbf{A}$ such that $\max_{1 \leq i \leq p}(\sum_{a \in A_i} l(a)) \leq D$?

Given an instance of the multiprocessor scheduling problem, we construct an instance of the load-balancing problem (shown in Fig. 8) in which 1) each s_i, $1 \leq i \leq n$, is an overloaded node with indivisible excess load of $l(a_i)$ units, and t is an underloaded node with deficit load of $\sum_{i=1}^{n} l(a_i)$ units; 2) there are p node-disjoint paths from u to v, all the edges on these paths have a capacity D, all the other edges have an infinite capacity, and $B = \sum_{i=1}^{n} l(a_i)$. Note that the construction can be done in polynomial time.

It is easy to see that at least B units of load imbalance can be eliminated without violating the link capacity and load transfer constraints if and only if there exists a solution for the multiprocessor scheduling problem. Since the multiprocessor scheduling problem is NP-complete in the strong sense, the decision version of the load-balancing problem with indivisible excess load is also NP-complete in the strong sense. □

Since it is unlikely to find a polynomial time optimal algorithm for the load-balancing problem with indivisible excess load, we propose below a heuristic algorithm for the problem. Let $G = (V, E)$ be the graph representation of the multicomputer or distributed system under consideration and $C(u, v)$ be the capacity (for load transferring purpose) of edge (u, v), for all $(u, v) \in E$. Let s_i, $1 \leq i \leq p$, be the overloaded nodes and t_i, $1 \leq i \leq q$, be the underloaded nodes. Each overloaded node s_i has indivisible excess load e_i, which must be routed to an underloaded node as an entity, and each underloaded node t_i has deficit load d_i, which is the maximum amount of load it can receive from overloaded nodes. Without loss of generality, we assume that e_is are sorted in nonincreasing order, i.e., $e_1 \geq e_2 \geq \cdots \geq e_p$.

The heuristic algorithm (see Fig. 9) consists of two phases. In Phase I, we treat the excess load as if it were divisible and use the network flow technique described in Section 3 to find a minimax flow f. If the excess load was indeed divisible, f would be an optimal solution in which the value $|f|$ is the maximum load imbalance that can be eliminated with the maximum link flow minimized. In Phase II, we use the minimax flow f found in Phase I as a "template" and route the entities of excess load one by one in such a way that the resulting flow on each link will be as close to the corresponding minimax flow as possible, i.e., the value $f(u, v)$ found in Phase I serves as the target flow for edge (u, v) to be achieved in Phase II. Since, in general, larger amounts of excess load are more difficult to route than smaller amounts, we will route the excess load in nonincreasing order of load amount.

We use e_i to refer to either the entity of excess load or its amount. During the execution of Phase II, $f'(u, v)$ is the total load currently routed through edge (u, v). When the excess load e_i is currently being routed, we say that an edge (u, v) is *feasible* if $C'(u, v) - f'(u, v) \geq e_i$ and a path from s_i to t is feasible if all edges on the path are feasible. We will route excess load e_i from the overloaded node s_i to an underloaded node t_j (actually, to node t) only via a feasible path, i.e., excess load can only be routed via a path in which each edge has enough (remaining) capacity. Note that, in Phase II, vertex s and edges (s, s_i), $1 \leq i \leq p$, are, in fact, not used, i.e., the underlying graph is $G'' = (V'', E'')$, where $V'' = V \cup \{t\}$ and $E'' = E \cup \{(t_i, t) \mid 1 \leq i \leq q\}$.

The excess load e_i is routed using a greedy type algorithm, called *ordered depth-first search* (O-DFS). Note that $f(u, v) - f'(u, v)$ is the difference between the target flow $f(u, v)$ and the total load $f'(u, v)$ currently routed through edge (u, v). A large $f(u, v) - f'(u, v)$ value implies that the current load routed through edge (u, v) is still far from the target value (note that $f(u, v) - f'(u, v)$ may be negative). Therefore, at each vertex u, we always choose to traverse next the edge (u, v) that has the largest $f(u, \cdot) - f'(u, \cdot)$ value among all feasible outgoing edges at

```
Algorithm INDIV-MMC01
Phase I.

Step 1. Construct a network $N = (V', E', s, t, C')$, where $V' = V \cup \{s, t\}$, $E' = E \cup \{(s, s_i) \mid 1 \le i \le p\} \cup \{(t_i, t) \mid 1 \le i \le q\}$, $C'(u, v) = C(u, v)$, for $(u, v) \in E$, $C'(s, s_i) = e_i$, for $1 \le i \le p$, and $C'(t_i, t) = d_i$, for $1 \le i \le q$.
        Let $w(u, v) = 1$, for $(u, v) \in E$, and $w(u, v) = 0$, otherwise.

Step 2. Treat each excess load $e_i$ as a divisible load, and use Algorithm MMC01 to find a minimax flow $f$ for $N$.

Phase II.

Step 1. Sort all overloaded nodes in nonincreasing order and assume that $e_1 \ge e_2 \ge \cdots \ge e_p$.
        Set $f'(u, v) := 0$, for all $(u, v) \in E'$.

Step 2. For $i \leftarrow 1$ to $p$ do the following:

    Step 2.1. Use the ordered depth-first-search (O-DFS) algorithm to find a feasible path from $s_i$ to $t$, where the ordered DFS algorithm is similar to the DFS graph traversal algorithm [13], except that when branching out from a node $u$ we always choose to traverse next the edge that has the largest $f(u, \cdot) - f'(u, \cdot)$ value among all untraversed feasible edges at node $u$.

    Step 2.2. If there does not exist any feasible path from $s_i$ to $t$, it means that the excess load $e_i$ will not be eliminated when the next round of load balancing is performed. If there exists a feasible path from $s_i$ to $t$, let $P$ be the (first) path found by the O-DFS algorithm ($P$ will be used as the route to eliminate the excess load $e_i$ when the next round of load balancing is performed). Reset $f'(u, v) := f'(u, v) + e_i$ for each edge $(u, v)$ on $P$.
```

Fig. 9. A heuristic algorithm for the case that excess load is indivisible.

u and, hence, reduce the maximum $f(u, \cdot) - f'(u, \cdot)$ value at node u.

We use the following example to further illustrate the heuristic algorithm INDIV-MMC01.

Example 1. Suppose the constructed network $N = (V', E', s, t, C')$ of a system graph $G = (V, E)$ and the capacity C' and minimax flow f found at the end of Phase I are shown in Fig. 10a. The maximum edge cost (link traffic) shown in the figure is 4 (note that only edges in E are considered).

In Phase II, we initially set $f'(u, v) = 0$ for all $(u, v) \in E'$. We first route excess load $e_1 (= 7)$. Starting from vertex s_1, since $f(s_1, v_2) - f'(s_1, v_2) = 4 > f(s_1, v_1) - f'(s_1, v_1) = 3$, the O-DFS algorithm will first visit vertex v_2. Since (v_2, t_2) is the only feasible outgoing edge at vertex v_2, the next vertex visited is t_2. Then, vertex t is visited and a feasible path from s_1 to t is found for e_1, i.e. the path s_1, v_2, t_2, t. For each edge (u, v) on that path, we reset $f'(u, v) = f'(u, v) + e_1$ as shown in Fig. 10b.

We next route excess load $e_2 (= 5)$. Using the O-DFS algorithm, we find the feasible path s_2, v_1, t_1, t for e_2. Note that, at vertex s_2, although $f(s_2, t_1) - f'(s_2, t_1) = 4 > f(s_2, v_1) - f'(s_2, v_1) = 1$, we still choose edge (s_2, v_1) since edge (s_2, t_1) is not feasible ($C'(s_2, t_1) - f'(s_2, t_1) = 4 < e_2 = 5$). For each edge (u, v) on the path found for e_2, we reset $f'(u, v) = f'(u, v) + e_2$, as shown in Fig. 10c. The next excess load to be routed is $e_3 (= 3)$ and the path found for e_3 is s_3, v_2, t_3, t and, for each edge (u, v) on the path, we reset $f'(u, v) = f'(u, v) + e_3$ as shown in Fig. 10d.

Finally, we route excess load $e_4 (= 2)$. Starting from s_4, O-DFS first traverses edge (s_4, t_3). At vertex t_3, since there is no feasible outgoing edge, O-DFS backtracks to vertex s_4 and then traverses edge (s_4, v_2). At vertex v_2, since t_3 has been visited, O-DFS next traverses (v_2, t_1). At vertex t_1, both (t_1, t) and (t_1, t_2) are feasible. Since

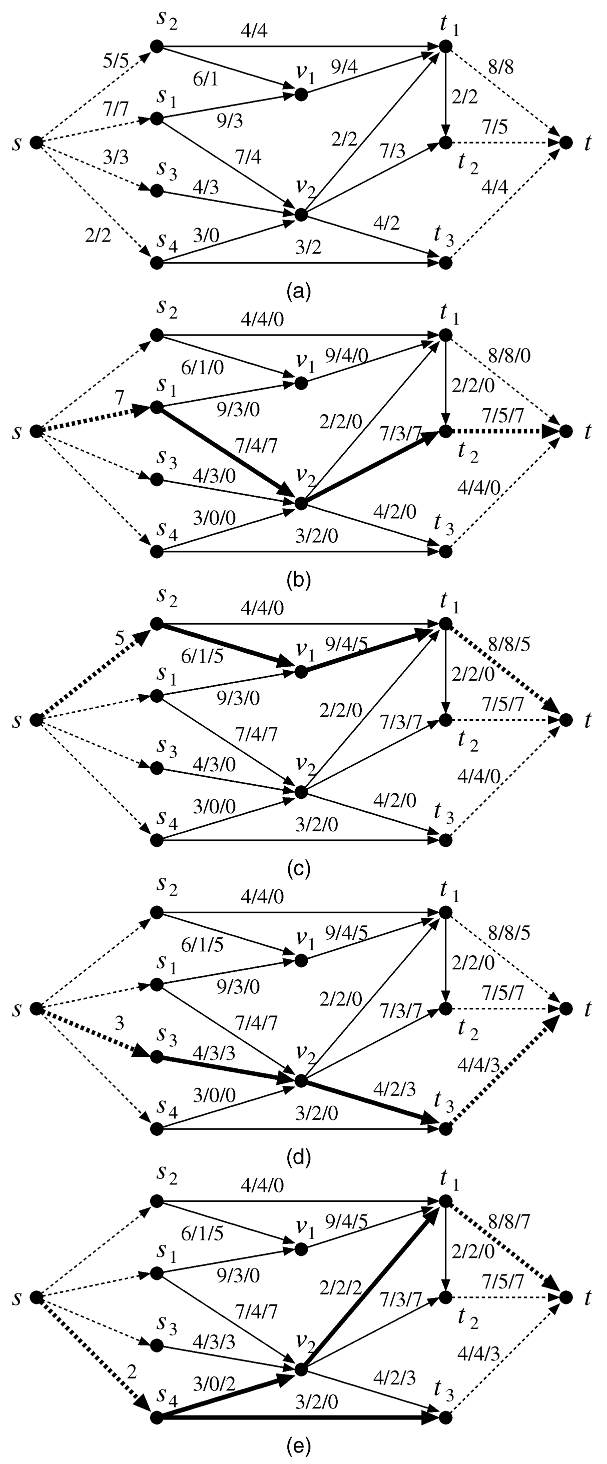


Fig. 10. An example that shows how the heuristic algorithm works. (a) After Phase I C'/f. (b) s_1 ($e_1 = 7$) $C'/f/f'$. (c) s_2 ($e_2 = 5$) $C'/f/f'$. (d) s_3 ($e_3 = 3$) $C'/f/f'$. (e) s_4 ($e_4 = 2$) $C'/f/f'$.

$f(t_1, t) - f'(t_1, t) = 3 > f(t_1, t_2) - f'(t_1, t_2) = 2$, the next edge traversed is (t_1, t) and the path found for e_4 is s_4, v_2, t_1, t. For each edge (u, v) on the path, we reset $f'(u, v) = f'(u, v) + e_4$ as shown in Fig. 10e.

The amount of load imbalance that can be eliminated in this example is $7 + 5 + 3 + 2 = 17$, and the maximum link traffic during load balancing is 7.

Note that the heuristic algorithm INDIV-MMC01 is not an optimal algorithm. It may not find the maximum load imbalance that can be eliminated and, in cases in which it does find the maximum load imbalance, it may not minimize the maximum link traffic.

It is difficult to implement an optimal algorithm for the load balancing problem with indivisible excess loads. However, the result of an optimal algorithm for indivisible excess loads cannot be better than that of Algorithm DIV-MMC01 for divisible excess loads. Our heuristic algorithm INDIV-MMC01 is compared against Algorithm DIV-MMC01 and two other heuristic algorithms, INDIV-Max-flow and INDIV-Simple by applying to an 8×8 mesh network used in Section 3 instead of comparing with an optimal algorithm. Algorithm INDIV-Maxflow is an algorithm using a maximum flow algorithm instead of MMC01 in Phase I of algorithm INDIV-MMC01. Algorithm INDIV-Simple has no Phase I and chooses the edge with largest $C'(u, .) - f(u, .)$ instead of that with largest $f(u, .) - f'(u, .)$ as the next traversing edge of node u in Phase II of Algorithm INDIV-MMC01.

In the mesh, every overloaded node has a total excess load of 800, which is divided into several entities. The number of entities in each overloaded node was varied from one to 16 and we experimented with three algorithms. Fig. 11 shows the average maximum link flows obtained from three algorithms. The maximum link flows obtained in two other heuristic algorithms get closer to the link capacity (= 1,000) as the number of entities in each overloaded node increases. By contrast, the maximum link flow in our heuristic algorithm INDIV-MMC01 is much smaller than the link capacity and gets closer to the optimal maximum link flow (= 640) obtained by DIV-MMC01 as the number of entities increases; in other words, the average size of entities decreases.

The time complexity of the heuristic algorithm is shown in the following theorem.

Theorem 3. *Phase II of Algorithm INDIV-MMC01 has a worst-case time complexity of $O(p \cdot m \cdot \log m)$, where p is the number of excess load entities. Algorithm Indiv-MMC01 has a worst-case time complexity of $O(m \cdot M(n, m) + p \cdot m \cdot \log m)$.*

Proof. As mentioned earlier, the underlying graph in finding paths from overloaded nodes to underloaded nodes in Phase II is $G'' = (V'', E'')$, where $V'' = V \cup \{t\}$, and $E'' = E \cup \{(t_i, t) \mid 1 \leq i \leq q\}$, where q is the number of underloaded nodes. The well-known DFS algorithm can be done in $O(x + y)$ time [13], [14], where x is the number of vertices and y is the number of edges of the graph traversed. For our O-DFS algorithm, each time when we first visit or backtrack to a vertex, we always traverse an untraversed edge with the maximum $f(\cdot, \cdot) - f'(\cdot, \cdot)$ value. Therefore, traversing all outgoing edges at a vertex u takes at most $O(d^o(u) \cdot \log d^o(u))$ time, where $d^o(u)$ is the out-degree of u and the logarithm is to the base 2. Thus, the total time to route excess load entities is at most [13], [14]

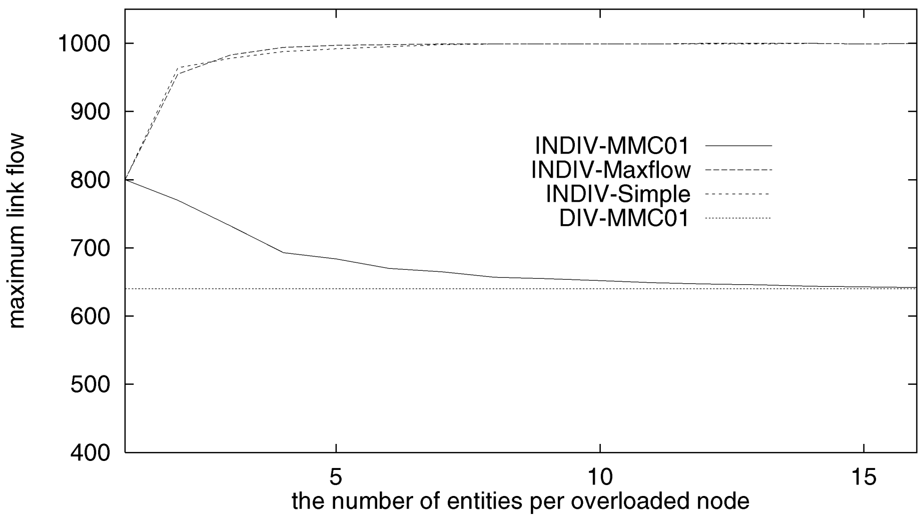


Fig. 11. Maximum link flows when excess load is indivisible.

$$|V''| + \sum_{v\in V''} d^o(v)\cdot \log d^o(v) \le |V''| + \log(m+q)\sum_{v\in V''} d^o(v)$$
$$= n+1+(m+q)\cdot\log(m+q).$$

Since we need to route p excess load entities, e_i, $1 \le i \le p$, the worst-case time complexity of Phase II is

$$O(p\cdot(n+1+(m+q)\cdot\log(m+q))) = O(p\cdot m\cdot\log m).$$

(Note that since we assume there is only one entity of indivisible excess load on each overloaded node and there is at least one directed path from a vertex to any other vertex in G, we have $m \ge n > p, q$.)

Phase I of Algorithm INDIV-MMC01 has a worst-case time complexity of $O(m\cdot M(n,m))$ since it is Algorithm DIV-MMC01 and its time complexity is $O(\ell\cdot M(n,m))$ and $1\le \ell\le m$, i.e., $\ell = O(m)$ in Section 2.2. Therefore, algorithm INDIV-MMC01 has a worst time complexity of $O(m\cdot M(n,m)+p\cdot m\cdot\log m)$. □

6 Conclusion

In this paper, we have considered the load-balancing problem in multicomputer/distributed systems that use circuit switching, wormhole routing, or virtual cut-through, with the objective of finding the maximum load imbalance that can be eliminated and the route to eliminate the load imbalance without violating the (traffic) capacity constraint on any link while minimizing the maximum link traffic among all links. Minimizing the maximum link traffic balances the network load among all links and reduces the probability of link contention.

We assume that load imbalances of each node can have arbitrary values, each link has a link traffic capacity, and the maximum traffic volume that can be transferred over it during load balancing. We also considered the effect of existing IPC traffic. We have solved the problem under various conditions. The solution approach is based on the minimax flow algorithm with a 0/1 weight function, MMC01. We gave an optimal algorithm DIV-MMC01 for the load-balancing problem with divisible excess loads with the time complexity of $O(m\cdot M(n,m))$, where n is the number of nodes/processors and m is the number of links in the system, and $M(x,y)$ is the time complexity of finding a maximum flow in a network of x vertices and y edges. Our algorithm assumes that the system does not have any specific routing schemes. We proposed a simple and unified graph transformation technique which transforms systems with specific routing schemes to systems without specific routing schemes, thus making our algorithm applicable to systems without specific routing schemes. We also considered the load-balancing problem for the case in which excess load is indivisible. We proved that the problem is NP-hard and proposed heuristic algorithm INDIV-MMC01 as a solution to the problem based on the DIV-MMC01 algorithm, with a time complexity $O(m\cdot M(n,m)+p\cdot m\cdot \log m)$ heuristic algorithm as a solution to the problem, where p is the number of excess load entities.

Acknowledgments

The work reported in this paper was supported in part by the US Office of Naval Research under Grant N00014-99-1-0465 and by the US National Science Foundation under Grant MIP-9203895 and by the KOSEF under the post-doctoral fellowship. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] L.M. Ni and P.K. McKinley, "A Survey of Wormhole Routing Techniques in Direct Networks," *Computer,* pp. 62-76, Feb. 1993.

[2] P. Kermani and L. Kleinrock, "Virtual Cut-Through: A New Computer Communication Switching Technique," *Computer Networks,* vol. 3, pp. 267-286, 1979.

[3] J. Upadhyay, V. Varavithya, and P. Mohapatra, "A Traffic-Balanced Adaptive Wormhole Routing Scheme for Two-Dimensional Meshes," *IEEE Trans. Computers,* vol. 46, no. 2, pp. 190-197, Feb. 1997.

[4] R.P. Bianchini and J.P. Shen, "Interprocessor Traffic Scheduling Algorithm for Multiple-Processor Networks," *IEEE Trans. Computers,* vol. 36, no. 4, pp. 396-409, Apr. 1987.

[5] D.D. Kandlur and K.G. Shin, "Traffic Routing for Multicomputer Networks with Virtual Cut-Through Capability," *IEEE Trans. Computers,* vol. 41, no. 10, pp. 1,257-1,270, Oct. 1992.

[6] S.H. Bokhari, "A Network Flow Model for Load Balancing in Circuit-Switched Multicomputers," *IEEE Trans. Parallel and Distributed Systems,* vol. 4, no. 6, pp. 649-657, June 1993.

[7] A.V. Goldberg and R.E. Tarjan, "A New Approach to the Maximum-Flow Problem," *J. ACM,* vol. 35, pp. 921-940, Oct. 1988.

[8] C.-C. Han, "A Fast Algorithm for the Minimax Flow Problem with 0/1 Weights," *Applied Math. Letters,* vol. 10, no. 2, pp. 11-16, 1997.

[9] R.E. Tarjan, *Data Structures and Network Algorithms.* Philadelphia: SIAM, 1983.

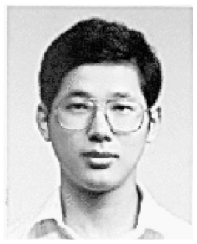
[10] R.K. Ahuja, "Algorithms for the Minimax Transportation Problem," *Naval Research Logistics Quarterly,* vol. 33, pp. 725-739, 1986.

[11] C. Glass and L.M. Ni, "The Turn Model for Adaptive Routing," *J. ACM,* vol. 41, pp. 874-902, Sept. 1994.

[12] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* San Francisco: W.H. Freeman, 1979.

[13] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms.* Cambridge, Mass.: The MIT Press, 1990.

[14] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms.* Reading, Mass.: Addison-Wesley, 1974.



Ching-Chih (Jason) Han received the BS degree in electrical engineering from National Taiwan University, Taiwan, Republic of China, in 1984, the MS degree in computer science from Purdue University, West Lafayette, Indiana, in 1988, and the PhD degree in computer science from the University of Illinois at Urbana-Champaign, in 1992. From August 1992 to January 1994, he was an associate professor in the Department of Applied Mathematics at National Sun Yat-sen University, Kaohsiung, Taiwan. From February 1994 to July 1996, he was a visiting associate research scientist in the Real-Time Computing Laboratory at the University of Michigan, Ann Arbor. From August 1996 to July 1997, he was an assistant professor in the Department of Electrical Engineering at The Ohio State University. From August 1997 to February 1999, he worked as a senior software engineer at BroadVision, Inc, Redwood City, California. Since February 1999, he has been CTO and cofounder of CreOsys, Inc, Fremont, California, a leading Internet solution provider which integrates real-time, Web-based, distributed graphic information services for the engineering world. His current research interests include Internet applications, real-time communications, parallel and distributed computing, and multimedia applications. He is a member of the IEEE Computer Society.

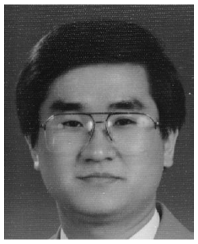
Kang G. Shin received the BS degree in electronics engineering from Seoul National University, Seoul, Korea in 1970, and the MS and PhD degrees in electrical engineering from Cornell University, Ithaca, New York, in 1976 and 1978, respectively. He is a professor and director of the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor.

He has authored/coauthored about 600 technical papers and numerous book chapters in the areas of distributed real-time computing and control, computer networking, fault-tolerant computing, and intelligent manufacturing. He coauthored (with C.M. Krishna) a textbook *Real-Time Systems* (McGraw-Hill, 1997). In 1987, he received the Outstanding IEEE Transactions on Automatic Control Paper Award and, in 1989, the Research Excellence Award from The University of Michigan. In 1985, he founded the Real-Time Computing Laboratory, where he and his colleagues are investigating various issues related to real-time and fault-tolerant computing.

His current research focuses on Quality of Service (QoS) sensitive computing and networking with emphases on timeliness and dependability. He has also been applying the basic research results to telecommunication and multimedia systems, intelligent transportation systems, embedded systems, and manufacturing applications.

From 1978 to 1982, he was on the faculty of Rensselaer Polytechnic Institute, Troy, New York. He has held visiting positions at the US Airforce Flight Dynamics Laboratory, AT&T Bell Laboratories, Computer Science Division within the Department of Electrical Engineering and Computer Science at the University of California at Berkeley, and International Computer Science Institute, Berkeley, California, IBM T.J. Watson Research Center, and Software Engineering Institute at Carnegie Mellon University. He also chaired the Computer Science and Engineering Division, Eletrical Engineering and Computer Science Department, The University of Michigan, for three years beginning in January 1991.

He is an IEEE fellow, was the program chairman of the 1986 IEEE Real-Time Systems Symposium (RTSS), the general chairman of the 1987 RTSS, the guest editor of the August 1987 special issue of *IEEE Transactions on Computers* on real-time systems, a program cochair for the 1992 International Conference on Parallel Processing, and served on numerous technical program committees. He also chaired the IEEE Technical Committee on Real-Time Systems during 1991-1993, was a distinguished visitor of the Computer Society of the IEEE, an editor of the *IEEE Transactions on Parallel and Distributed Computing*, and an area editor of the *International Journal of Time-Critical Computing Systems*.



Sang Kyun Yun received the BS degree in electronics engineering from Seoul National University, Korea, in 1984 and the MS and PhD degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST) in 1986 and 1995, respectively. He is currently an associate professor in the Department of Computer Science, Seowon University, Cheongju, Korea. He worked at Hyundai Electronics Industries, Korea, from 1986 to 1990. He was a visiting researcher in the Real-Time Computing Laboratory, the Department of Electrical Engineering and Computer Science at the University of Michigan, Ann Arbor, during 1998. His interests include interconnection network, parallel processing, computer architecture, computer network, and internet/web application. He is a member of the IEEE Computer Society.