The Design of a Dynamic Efficient Load Balancing Algorithm on Distributed Networks[*]

Yeojin Lee[1], Okbin Lee[1], Wankyoo Choi[1], Chunkyun Youn[2], and Ilyong Chung[1,**]

[1] Dept. of Computer Science and BK Team, Chosun University, Gwangju, Korea
iyc@chosun.ac.kr
[2] Department of Internet Software, Honam University, Gwangju, Korea
chqyoun@itc.honam.ac.kr

Abstract. In order to maintain load balancing in a distributed network, each node should obtain workload information from all the nodes in the network. To accomplish this, this processing requires $O(v^2)$ communication complexity, where v is the number of nodes. First, we present a new synchronous dynamic distributed load balancing algorithm on a $(v, k+1, 1)$-configured network applying a symmetric balanced incomplete block design, where $v = k^2 + k + 1$. Our algorithm designs a special adjacency matrix and then transforms it to $(v, k+1, 1)$-configured network for an efficient communication. It requires only $O(v\sqrt{v})$ communication complexity and each node receives workload information from all the nodes without redundancy since each link has the same amount of traffic for transferring workload information. Later, this algorithm is reconstructed on distributed networks, where v is an arbitrary number of nodes and is analyzed in terms of efficiency of load balancing.

1 Introduction

In a distributed network it is likely that some nodes are heavily loaded while others are lightly loaded or idle. It is desirable that workload be balanced between these nodes so that utilization of nodes can be increased and response time can be reduced. A load balancing scheme[1]-[3] determines whether a task should be executed locally or by a remote node. This decision can be made in a centralized or distributed manner. In a distributed network, distributed manner is recommended. In order to make this decision, each node can be informed about the workload information of other nodes. Also this information should be the latest because outdated information may cause an inconsistent view of the system state. So disseminating load information may incur a high link cost or a significant communication traffic overhead. For example, the ARPANET[4] routing algorithm is a distributed adaptive algorithm using estimated delay as

[*] This research was supported by the Program for the Training of Graduate Students in Regional Innovation which was conducted by the Ministry of Commerce Industry and Energy of the Korean Government.

[**] Corresponding author.

the performance criterion and a version of the backward-search algorithm[5]. For this algorithm, each node maintains a delay vector and a successor node vector. Periodically, each node exchanges its delay vector with all of its neighbors. On the basis of all incoming delay vectors, a node updates both of its vectors.

In order to decrease communication overhead for obtaining workload information from all the nodes in the network, messages should be exchanged between adjacent nodes and then load balancing process be performed periodically by using these local messages. So each processor balances the workload with its neighbors so that the whole system will be balanced after a number of iterations. CWA(Cube Walking Algorithm)[6] is employed for load balancing on hypercube network. It requires $O(v^2)$ communication complexity and a communication path is $O(log_2 v)$. To reduce communication cost, flooding scheme is applied. However, the overlap of workload information occurs[7]-[8]. Based on SBN(Symmetric Broadcast Networks), communication patterns between nodes are constructed. It also needs $O(v^2)$ communication complexity for collecting workload information from all the nodes and a communication path is $O(log_2 v)$[9]-[10].

In this paper we design the network topology consisting of v nodes and $v \times k$ links and each node of which is linked to $2k$ nodes, where $v = k^2 + k + 1$. On this network, each node receives information from k adjacent nodes and then sends these information to other k adjacent nodes periodically. So, each node receives workload information for $k^2 + k$ nodes with two-round message interchange. Our algorithm needs only $O(v\sqrt{v})$ communication complexity. Later, this algorithm is revised for distributed networks and is analyzed in terms of efficiency of load balancing.

2 About (v, k, λ)-Configuration

Let $V = \{0, 1, ..., v-1\}$ be a set of v elements. Let $B = \{B_0, B_1, ..., B_{b-1}\}$ be a set of b blocks, where B_i is a subset of V and $|B_i| = k$. For a finite incidence structure $\sigma = \{V, B\}$, if σ satisfies following conditions, then it is a balanced incomplete block design(BIBD)[11], which is called a (b, v, r, k, λ)-configuration.

1. B is a collection of b k-subsets of V and these k-subsets are called the blocks.
2. Each element of V appears exactly r of the b blocks.
3. Every two elements of V appears simultaneously in exactly λ of the b blocks.
4. $k < v$.

For a (b, v, r, k, λ)-configuration, if it satisfies $k = r$ and $b = v$, then it is a symmetric balanced incomplete block design (SBIBD)[12] and it is called a (v, k, λ)-configuration. There are some relations among parameters b, v, r, k, λ that are necessory conditions for existence of this configuration, $bk = vr$ and $r(k-1) = \lambda(v-1)$.

3 Generation of a $(v, k+1, 1)$-Configuration

We now present an algorithm to generate an incidence structure $\sigma = \{V, B\}$ satisfying the condition for a $(v, k+1, 1)$-configuration in the case that k is a prime number. This $(v, k+1, 1)$-configuration is employed for constructing network topology below.

3.1 Design of an Algorithm to Construct $(v, k+1, 1)$-Configuration

In order to construct $(v, k+1, 1)$-configuration, $(k+1)$ sectors are designed. The first sector is composed of $(k+1)$ blocks and the others k blocks. Each block in the first sector contains element 0. The remaining elements of the first block, the second block,..., the k^{th} block of the first sector are $(1, 2, ..., k)$,$(k+1, k+2, ..., 2k)$,...,$(k^2+1, k^2+2, ..., k^2+k)$,respectively. Each block in the i^{th} sector contains element $(i-1)$ and the remaining k elements are chosen from $(k+1, k+2, ..., k^2+k)$. According to Algorithm 1, incidence structure X generates the first sector and incidence structure Y the remaining sectors.

Algorithm 1 for Generating an incidence structure

Incidence structure $T = \{V, B\}$, where $V = \{0, 1,.., v-1\}$, $B = \{B_0, B_1,.., B_{b-1}\}$, $|B_i| = k+1$. $B_{i,j}$ is the j^{th} element of B_i

1. Select a prime number k and compute $v = k^2 + k + 1$.
2. Construct two incidence structures $X = \{V, C\}$ and $Y = \{V, D\}$.
   (a) $C_{i,j} = \begin{bmatrix} 0 & if\ j = 0 \\ & \\ t, t = i \times k + j & if\ j \geq 1 \end{bmatrix}$
   $0 \leq i, j \leq k.$
   (b) $D_{i,j} = \begin{bmatrix} C_{0,t}\ , t = \lfloor i/k \rfloor + 1 & if\ j = 0 \\ & \\ C_{j,t}\ , t = 1 + (i + (j-1) \times \lfloor i/k \rfloor)\ mod\ k & if\ j \geq 1 \end{bmatrix}$
   $0 \leq i \leq (k^2 - 1),\ 0 \leq j \leq k.$
3. Generate $Z = \{V, B\}$ from X and Y.
   $B_i \longleftarrow C_i$
   $B_{i+k+1} \longleftarrow D_i$

The table below illustrates how to create $Z = \{V, B\}$, $V = \{0, 1, ..., 12\}$. We now prove that this structure satisfies the conditions of a $(v, k+1, 1)$-configuration.

Definition 1. On incidence structure Y, Sector S_i is the i^{th} family of k blocks, $D_j \in S_i$, $i = \lfloor j/k \rfloor$.

For example, If k equals 3 , then $\lfloor 0/k \rfloor = \lfloor 1/k \rfloor = \lfloor 2/k \rfloor = 0$. So, $S_0 = \{D_0, D_1, D_2\}$. There are k sectors in Y.

Lemma 1. For two elements $D_{i1,j1}$ and $D_{i2,j2}$, $D_{i1,j1} \neq D_{i2,j2}$, $if\ j1 \neq j2$.
Proof: From Algorithm 1-2-(a), if $0 < j \leq k,\ 0 \leq i \leq k$ then $C_{i,j} = i \times k + j$.

Table 1. A set of blocks on Z generated from Algorithm 1

X
C_0= { 0, 1, 2, 3 }
C_1= { 0, 4, 5, 6 }
C_2= { 0, 7, 8, 9 }
C_3= { 0, 10, 11, 12 }

Y
D_0= { 1, 4, 7, 10 }
D_1= { 1, 5, 8, 11 }
D_2= { 1, 6, 9, 12 }
D_3= { 2, 4, 8, 12 }
D_4= { 2, 5, 9, 10 }
D_5= { 2, 6, 7, 11 }
D_6= { 3, 4, 9, 11 }
D_7= { 3, 5, 7, 12 }
D_8= { 3, 6, 8, 10 }

$\Longrightarrow$

Z
B_0= { 0, 1, 2, 3 }
B_1= { 0, 4, 5, 6 }
B_2= { 0, 7, 8, 9 }
B_3= { 0, 10, 11, 12 }
B_4= { 1, 4, 7, 10 }
B_5= { 1, 5, 8, 11 }
B_6= { 1, 6, 9, 12 }
B_7= { 2, 4, 8, 12 }
B_8= { 2, 5, 9, 10 }
B_9= { 2, 6, 7, 11 }
B_{10}= { 3, 4, 9, 11 }
B_{11}= { 3, 5, 7, 12 }
B_{12}= { 3, 6, 8, 10 }

This means if $j > 0$ then all the elements are distinct. And as shown in Algorithm 1-2-(b), an element of C_j is placed on the j^{th} element of a certain block of Y if $D_{i,j} = C_{j,t}, t \neq 0$.

Lemma 2. For a sector consisting of k blocks, the first element of each block has the same value and the other k^2 elements are equal to $V - C_0$.
Proof: In the case that $D_{i,0} = C_{0,\lfloor i/k \rfloor +1}$, the first element of k blocks on a sector has the same value. According to Algorithm 1-2-(b), $D_{i,j} = C_{j,t}, t = 1 + (i + (j-1)\lfloor i/k \rfloor)\ mod\ k$. Since k is a prime number, each element except the first element of each block is distinct and these distinct k^2 elements are equal to $V - C_0$.

Lemma 3. For incidence structure Y, $D_{a,j} = D_{b,j}, j \geq 1$, if
$b = ((a - c(j-1))\ mod\ k\ + k(\lfloor a/k \rfloor + c))\ mod\ k^2$.
Proof: From Algorithm 1-2-(b), $D_{a,j} = C_{j,t}$. We now prove that $D_{b,j} = C_{j,t}$. t can be calculated from parameters b, j below. Then t obtained on this lemma is equal to that from Algorithm 1-2-(b). Therefore, $D_{a,j} = D_{b,j}$.

$$\begin{aligned} t &= 1 + (b + (j-1) \times \lfloor b/k \rfloor)\ mod\ k \\ &= 1 + (((a - c(j-1))\ mod\ k\ + k(\lfloor a/k \rfloor + c)) + (j-1)\lfloor((a - c(j-1))\ mod\ k\ + \\ &\quad k(\lfloor a/k \rfloor + c))/k\rfloor)\ mod\ k \\ &= 1 + (a - c(j-1)) + (j-1) \times (\lfloor a/k \rfloor + c)\ mod\ k \\ &= 1 + (a + (j-1)\lfloor a/k \rfloor)\ mod\ k \end{aligned}$$

Here, if $D_{a,j}$ is in sector S_s then $D_{b,j}$ is in $S_{(s+c)\ mod\ k}$. In case of $c \equiv 0\ (mod\ k)$, then $a = b$.

Lemma 4. Each element of V appears in exactly $k + 1$ times in Z.
Proof: According to Algorithm 1-2-(a), $C_{i,0} = 0$. Since $0 \leq i \leq k$, 0 appears $k + 1$ times. The other $v - 1$ elements, $V - \{0\}$, appear exactly once on X. From Lemma 3, each element of $C_{0,j}, 1 \leq j \leq k$, appears k times in a sector of Y and the rest k^2 elements appear once in every sector of Y. Therefore, each element appears $k + 1$ times in Z.

Lemma 5. Any pair of elements of V appears in exactly only once in Z.

Proof: The first element of V makes a pair with all the other elements and this pair appears once by designing rule of incidence structure(see Algorithm 1-2-(a)). Each element of $C_{0,j}, 1 \leq j \leq k$ makes a pair with $V - C_0$ elements and it also appears once proven by Lemma 3. The rest k^2 elements are now considered. For an arbitrary pair $D_{a,j1} = D_{a,j2}$, $j1, j2 \geq 1$, in order to make the same pair on other block D_b, the two elements should be on the same block. According to Lemma 4, if $j1 = j2$, then they are located on D_b. However, this case does not occur since $j1 \neq j2$. Therefore, any pair of elements of V appears in exactly once in Z.

Theorem 1. Z designed by Algorithm 1 satisfies the conditions of a $(v, k+1, 1)$-configuration.

Proof: Z satisfies the conditions of the SBIBD by Lemma 4 and Lemma 5.

3.2 Design of Network Configuration

In order to construct a network topology which has minimum link cost and traffic overhead, we imported $(v, k+1, 1)$-configuration. An incidence structure $Z = \{V, B\}$ satifies the conditions for a $(v, k+1, 1)$-configuration and M is a binary incidence matrix of Z . Then this matrix M can be transformed to an adjacency matrix of a graph $G = \{V, E\}$. Based on this idea, network topology can be designed as follows.

Algorithm 2 for Design of Network Configuration

1. Create an incidence structure $Z = \{V, B\}$ by Algorithm 1.
2. Generate $L = \{V, E\}$ from Z by exchanging blocks so that each block E_i includes element i.

```
$E_0 \longleftarrow B_0$
for ( $i = 1$ ; $i \leq k$ ; $i = i + 1$ )
    $E_{(i+1)k} \longleftarrow B_i$
for ( $i = k + 1$ ; $i < v$ ; $i = i + 1$ ) {
    if ( $(B_{i,\lceil i/k \rceil - 1}$ $mod$ $k) = 0$) then
        $E_{\lceil i/k \rceil - 1} \longleftarrow B_i$
    else {
        if ( $(i$ $mod$ $k) = 0$) then   $t = i - k$
      else   $t = \lfloor i/k \rfloor * k$
        $E_{t+(i \ mod \ k)} \longleftarrow B_i$ }
}
```

3. Create an adjacency matrix $A = (a_{i,j})$ for graph G from L , where G is a network topology containing v nodes.

$$a_{i,j} = \begin{bmatrix} 1 & if \ (i \neq j) \ and \ (j \in E_i) \\ 0 & otherwise \end{bmatrix}$$

Table 2. Blocks of L generated from Z of Table 1

L
$E_0 = \{ 0, 1, 2, 3 \}$
$E_1 = \{ 1, 6, 9, 12 \}$
$E_2 = \{ 2, 5, 9, 10 \}$
$E_3 = \{ 3, 5, 7, 12 \}$
$E_4 = \{ 1, 4, 7, 10 \}$
$E_5 = \{ 1, 5, 8, 11 \}$
$E_6 = \{ 0, 4, 5, 6 \}$
$E_7 = \{ 2, 6, 7, 11 \}$
$E_8 = \{ 2, 4, 8, 12 \}$
$E_9 = \{ 0, 7, 8, 9 \}$
$E_{10} = \{ 3, 6, 8, 10 \}$
$E_{11} = \{ 3, 4, 9, 11 \}$
$E_{12} = \{ 0, 10, 11, 12 \}$

Lemma 6. The ith row of the adjacency matrix obtained from Algorithm 2 contains element i.
Proof: In Fig. 1, $(k+1)$ sectors are described - one sector on X and k sectors on Y. Each sector on X and Y is composed of $(k+1)$ and k blocks, respectively. New sectors on Z' would be generated. Since the ith block on $X(i > 0)$ includes element $(i+1)k$, it is relocated on S_i on Z' - $E_0, E_{2k}, E_{3k}, ..., E_{(k+1)k}$. We now look into S_i on Z, $1 \leq i \leq k$. $(k-1)$ blocks of it can be placed on S_i on Z' and the remainder on the ith block of S_0 on Z' because B_{ik+j}, $1 \leq j \leq k$, the jth block of S_i on Z, includes one of $\{ik+1, ik+2, ..., ik+(k-1), i\}$. If a certain block contains $(ik+2)$, then it is placed on E_{ik+2}. Therefore, the ith block on Z' can include element i.

G has v nodes since G is created from $(v, k+1, 1)$-configuration. Each block E_i is composed of $k+1$ elements including i. Each node obtains $2k$ links from Step 3 of Algorithm 2. So, G becomes a 2k-regular graph. Therefore there are $(2k \times v)/2 = vk$ links in G. Given $Z = \{V, B\}$ obtained from Algorithm 1, performance of Algorithm 2 is shown on Table 2.

4 Design of an Efficient Load Balancing Algorithm on Distributed Networks

An efficient load balancing algorithm is now constructed on $(v, k+1, 1)$-configured networks generated by Algorithm 2.

Definition 2. Construct two sets P_i and R_i consisting of adjacent k nodes, where P_i is a set of nodes to send workload information to node i at time T_{2t}, and R_i is a set of nodes to send workload information to node i at time T_{2t+1}.
$P_i = \{j \mid j \in E_i - \{i\}\}$
$R_i = \{j \mid i \in E_j, (0 \leq j \leq n-1) \textit{ and } (i \neq j)\}$

The following example will provide a better understanding of the concept given by Definition 2. Nodes $3, 6, 9$ and node 12 send their workload information to node 1 at time T_{2t}, and P_1 is {3,6,9,12}, while nodes $0, 4$ and node 5 send the information to node 1 at time T_{2t+1}, and R_1 is {0,4,5}.

Definition 3. Generate two sets SF_i and RF_i, where $SF_i(j)$ is a set of workload information transmitted from node j to node i at time T_{2t} and $RF_j(i)$ is workload information transmitted from node j to node i at time T_{2t+1}.
$SF_i = \{SF_i(j) \mid j \in P_i, SF_i(j) = j\}$.
$RF_i = \{RF_j(i) \mid j \in R_i, RF_j(i) = \{E_j - \{i\}\}$.

Algorithm 3 for Construction of an Efficient Load Balancing Algorithm on (v,k+1,1)-configured networks

1. Node i receives a set of workload information $SF_i(j)$ from node $j \in P_i$ at T_{2t} and renews a table of workload information.
2. Node i receives a set of workload information $RF_j(i)$ from node $j \in R_i$ at T_{2t+1} and renews a table of workload information.
3. Repeat the first step.

The following table indicates that node i receives workload information $SF_i(j)$ and $RF_j(i)$ from node j at times T_{2t} and T_{2t+1}, respectively. For example, node 1 receives information on {6}, {9} and {12} from $SF_1(6)$, $SF_1(9)$ and $SF_1(12)$ at T_{2t}, and receives information on {0,2,3}, {4,7,10} and {5,8,11} from $RF_0(1)$, $RF_4(1)$ and $RF_5(1)$ at T_{2t+1}. Therefore, node 1 can obtain workload information for all the nodes at T_{2t+2} and this fact is proven in Theorem 2.

Theorem 2. According to Algorithm 3, every node obtains workload information for all the nodes at T_{2t+2}.
Proof: At T_{2t}, node j sends workload information $SF_i(j)$ to node i and then node i receives k workload information. At T_{2t+1}, node i receives workload information from node j by Algorithm 3-2. On an arbitrary pair ($RF_{i1}(j)$, $RF_{i2}(j)$) ,$i1 \neq i2$, intersection of these sets is empty since on $(v, k+1, 1)$-configuration, every pair of two objects appears simultaneously in exactly one of v blocks and node j is an element of R_{i1} and R_{i2}, respectively. So node i obtains workload information for k^2 nodes at T_{2t+1}. Therefore, node i receives workload information for $k^2 + k$ nodes at T_{2t+2}.

In case that $v1 \neq k^2 + k + 1$, $(v - v1)$ nodes are deleted starting from node v node to node $v1$. The load balancing algorithm with $v1$ nodes is the same as Algorithm 3 except node i receives information from node j, where $i, j < v1$.

5 Analysis of an Efficient Load Balancing Algorithm

Algorithm 3 can be performed well when $v = k^2 + k + 1$. Each node receives workload information from all the nodes in $O(v\sqrt{v})$ time. However, the number of nodes may not be v in real-world application. Then we can not gurantee a desired result for an efficient load balancing. The algorithm now is analyzed in terms of how much a node receives information.

Definition 3. w_0 is the number of information on missing nodes for $S_{\lceil n/k \rceil}$ and w_1 from $S_{\lceil n/k \rceil+1}$ to the S_{k+1}.

Lemma 7. If $(n \ mod \ k) = 0$ then $w_0 \ = \ _kP_2$. Otherwise, $w_0 \ = \ k2 *_{k1} P_2 +_{k-k2} P_2$, where $k1 = n/k$ and $k2 = (k * (k1 + 1) - n)$.
Proof: In case that $n = (i+1)k$, $E_n = (0, ik+1, ik+2, ..., (i+1)k)$. Suppose that node n is deleted. According to Algorithm 3, node 0 does not receive workload information on (k-1) nodes - node $(ik+1)$, node $(ik+2)$,..., node $((i+1)k-1)$. Neither the other $(k-1)$ nodes in E_n do. So w_0, the number of information on missing nodes, is $k*(k-1)$. In case that $n \neq (i+1)k$ and node n is deleted. $(k2+1)$ nodes are deleted from $E_{\lceil n/k \rceil * k}$. $(k-k2)$ nodes do not receive information. Then the number of missing information is $_{k-k2}P_2$. Because of deleting node n from E_n, $k1$ nodes do not receive information and the number of missing information is $_{k1}P_2$. $(k2-1)$ blocks - $E_{n+1}, E_{n+2}, ..., E_{\lceil n/k \rceil * k-1}$ are the same as E_n and the number of missing information for these blocks is $(k2-1) *_{k1} P_2$. Therefore, $w_0 \ = \ k2 *_{k1} P_2 +_{k-k2} P_2$.

Lemma 8. If $(n \ mod \ k) = 0$ then $w_1 = (k - k1 + 1) * ((k-2) *_{k1} P_2 +_{k1-1} P_2)$. Otherwise, $w_1 \ = \ (k - k1) * ((k - k3 - 2) *_{k1+1} P_2 + (k3+1) *_{k1} P_2)$, where if $(n \leq E_{(k1+1),k1})$ then $k3 = k2 - 1$ else $k3 = k2$.
Proof: In case that $(n \ mod \ k) = 0$ and node n is deleted. The number of n is $(k - k1 + 1)$ in $(k - k1 + 1)$ sectors - from S_{k1+1} to S_{k+1}. In other word, a sector consisting of $(k-1)$ blocks has one n. For a block deleting n, $(k1-1)$ nodes do not receive information and the number of missing information is $_{k1-1}P_2$ and for the remaining $(k-2)$ blocks, $k1$ nodes do not receive information and the number of missing information is $_{k1}P_2$. Since w_1 for a sector is $(k-2) *_{k1} P_2 +_{k1-1} P_2$, w_1 for $(k - k1 + 1)$ sectors is $(k - k1 + 1) * ((k-2) *_{k1} P_2 +_{k1-1} P_2)$. In case that $n \neq (i+1)k$ and node n is deleted. Similarly, the number of n is $(k - k1)$ in $(k - k1)$ sectors - from the S_{k1} to the S_{k+1} sector. For a block deleting $m, n \leq m \leq k * (n/k + 1)$, $k1$ nodes do not receive information and the number of missing information is $_{k1}P_2$. Because the number of blocks deleting m in a sector is $(k2+1)$, the number of missing information is $(k2+1) *_{k1} P_2$. For the $(k - k2 - 2)$ remaining blocks, $(k1+1)$ nodes do not receive information and that is $(k - k2 - 2) *_{k1+1} P_2$. One important thing must be considered. According to Algorithm 2, a certain block in Sector $i, 1 \leq i \leq k$ moves to the E_i - the i^{th} block in S_0. $E_{i,i-1}$ does not appear in S_i. Where $i = k1 + 1$, if $(n \leq E_{(k1+1),k1})$ then $k3 = k2 - 1$ else $k3 = k2$. Therefore, w_1 is $(k - k1) * ((k - k3 - 2) *_{k1+1} P_2 + (k3+1) *_{k1} P_2)$.

Algorithm 4 for Computing the number of information on missing nodes given n nodes

```
$n \neq k^2 + k + 1$
$k1 = n/k$
$k2 = (k * (k1 + 1) - n)$
if ( $(n \ mod \ k) = 0$) then {
    $w_0 \ = \ {}_kP_2$
    $w_1 \ = (k - k1 + 1) * ((k - 2) *_{k1} P_2 +_{k1-1} P_2)$
    }
    else {
if $(n \leq E_{(k1+1),k1})$ then
    $k3 = k2 - 1$
    else $k3 = k2$
    $w_0 \ = \ k2 *_{k1} P_2 +_{k-k2} P_2$
    $w_1 \ = \ (k - k1) * ((k - k3 - 2) *_{k1+1} P_2 + (k3 + 1) *_{k1} P_2)$
    }
$w \ = \ w_0 + w_1$
```

Given the number of nodes - from 43 nodes to 32 nodes, the following table illustrates the result of executing Algorithm 4.

Table 3. The number of Information on missing nodes required for load balancing

the number of nodes	w_0	w_1
43	$6 *_6 P_2 +_1 P_2$	$6 *_6 P_2$
42	${}_7P_2$	$2 * (5 *_6 P_2 + 1 *_5 P_2)$
41	$1 *_5 P_2 +_6 P_2$	$2 * (4 *_6 P_2 + 2 *_5 P_2)$
40	$2 *_5 P_2 +_5 P_2$	$2 * (3 *_6 P_2 + 3 *_5 P_2)$
39	$3 *_5 P_2 +_4 P_2$	$2 * (2 *_6 P_2 + 4 *_5 P_2)$
38	$4 *_5 P_2 +_3 P_2$	$2 * (1 *_6 P_2 + 5 *_5 P_2)$
37	$5 *_5 P_2 +_2 P_2$	$2 * (1 *_6 P_2 + 5 *_5 P_2)$
36	$6 *_5 P_2 +_1 P_2$	$2 * (0 *_6 P_2 + 6 *_5 P_2)$
35	${}_7P_2$	$3 * (5 *_5 P_2 + 1 *_4 P_2)$
34	$1 *_4 P_2 +_6 P_2$	$3 * (4 *_5 P_2 + 2 *_4 P_2)$
33	$2 *_4 P_2 +_5 P_2$	$3 * (3 *_5 P_2 + 3 *_4 P_2)$
32	$3 *_4 P_2 +_4 P_2$	$3 * (2 *_5 P_2 + 4 *_4 P_2)$

6 Conclusion

Researches have shown that distributed load balancing schemes can reduce task turnaround time substantially by performing tasks at lightly loaded nodes in the network. However, maintaining workload information on all the nodes is necessary for such scheme, while it may incur large communication overhead. In this paper, in order for the system to increase utilization and to reduce response time, we present an efficient load balancing algorithm on $(v, k+1, 1)$-configured networks consisting of v nodes and vk links by employing the symmetric balanced incomplete block design. To accomplish this, each node should receive workload information from all the nodes without redundancy and each link should have the same amount of traffic for transferring workload information. Our algorithm requires two rounds of message interchange and $O(v\sqrt{v})$ communication complexity, as opposed to $O(v^2)$ communication complexity needed by protocols which use only one round of message interchange. Later, this algorithm is reconstructed on distributed networks, where v is an arbitrary number of nodes and is analyzed in terms of efficiency of load balancing.

References

1. C.Hui, S.Chanson, Hydrodynamic Load Balancing, IEEE Transactions on Parallel and Distributed System, vol. 10, no. 11, pp. 1118-1137, 1999.
2. V. Bharadwaj, D. Ghose and T. Robertazzi, "A New Paradigm for Load Scheduling in Distributed Systems," Cluster Computing, vol. 6, no. 1, pp. 7-18, 2003.
3. A. Legrand, et al., "Mapping and Load-Balancing Iterative Computing," IEEE Transactions on Parallel and Distributed System, vol. 15, no. 6, pp. 546-558, 2004.
4. M. Padlipsky, A perspective on the ARPANET Reference Model, Proc. of INFOCOM, IEEE, 1983.
5. L. Ford, D. Fulkerson, Flow in Network, Princeton University Press, 1962.
6. M. Wu, On Runtime Parallel Scheduling for processor Load balancing, IEEE Transactions on Parallel and Distributed System, vol. 8, no. 2, pp. 173-185, 1997.
7. K. Nam, J. Seo, Synchronous Load balancing in Hypercube Multicomputers with Faulty Nodes, Journal of Parallel and Distributed Computing, vol. 58, pp. 26-43, 1999.
8. H. Rim, J. Jang, S. Kim, Method for Maximal Utilization of Idle links for Fast Load Balancing, Journal of Korea Information Science Society, vol. 28, no. 12, pp. 632-641, 2001.
9. S. Das, D. Harvey, and R. Biswas, Adaptive Load-Balancing Algorithms Using Symmetric Broadcast Networks, Journal of parallel and Distributed Computing, vol. 62, no. 6, pp. 1042-1068, 2002.
10. S. Das, D. Harvey, and R. Biswas, Parallel Processing of Adaptive Meshes with Load Balancing, IEEE Transactions on Parallel and Distributed Systems, vol. 12, no. 12, pp. 1269-1280, 2001.
11. C.L.Liu, Introduction to Combinatorial Mathematics, pp. 359-383, McGraw-Hill, 1968.
12. I. Chung, W. Choi, Y. Kim, M. Lee, The Design of conference key distribution system employing a symmetric balanced incomplete block design, Information Processing Letters, vol. 81, no. 6, pp. 313-318, 2002.3.