Proposed Global Centralized Dynamic Load Balancing Combining with Shortest Routing in Hyper-De Bruijn-Aster of Multiprocessors Network

Pham Minh Tri, Nguyen Hong Thai, Nguyen Chi Ngoc
Dept. of Telecommunication
Ho Chi Minh City University of Technology, South Vietnam
minhtriphamlth@yahoo.com, nhthai2005@gmail.com, ncngoc@hcmut.edu.vn

Abstract

Efficient Dynamic Load Balancing algorithms are the key to many efficient parallel applications. Until now, research in this area mainly focused on the general methods which are hardly applied in specific frameworks. In this paper, we proposed an identified model of Global Centralized Dynamic Load Balancing adapting to the random changes of network. Via the essential characteristics of Hyper-De Bruijn –Aster Graph [8], we set the static routing rule protocol in order to minimize the time execution and the complexity of communication.

By combining with the Shortest Routing in Hyper-De Bruijn-Aster Network and assigning to each processor work proportional to its performance, we present a simply strategy of Global Centralized Dynamic Load Balancing which provides the high scalability, the easier monitoring and management of Multiprocessors Network.

1 Introduction

Arrangement, programming etc. of multiprocessors network are investigated and developed effectively (for instance, [5], [6], [7]), but one of the optimal Graphs, Hyper-De Bruijn-Aster [HD*] becomes the attractive Graph which is extremely efficient implementation in Dynamic Load Balancing Multiprocessors Network.

HD* provides the maximal desired Fault Tolerant; the optimal point to point Routing algorithm and Broadcasting in Multiprocessors Network. Thus, an important approach in this paper is the rule (i.e., routing rule or firing rule) of Shortest Routing in HD*, which is simply and effective. In HD*, we can also compute the amount of nodes (consists of their address) between source node A and destination node B in which the packets passed on. To find out the features of HD*, see [8] for more detailed study.

With Multiprocessors Network, the solution of Dynamic Load Balancing problem is desirable. Load Balancing involves assigning the tasks to each processor in proportion to its performance, this would minimize the execution time (or cost of processing) of the program. In fact, Static Load Balancing strategy is suitable for ideal environment of network (i.e., the network of the same physical architecture of multiprocessors) in which all of pro-cessors are homogeneous in during of run time or in normal state and this strategy also applied in Oriented Routing of communication atmosphere. However, many parallel applications produce work load dynamically and its amount per processors often changes dramatically during run time. Therefore, most of processors usually cannot achieve the ideal conditions, for example: the

processors speed and the capacity of storing tasks usually change at different executive time. Routing algorithms must always match with the changes of network, and they are frequently updated in order to adapt the varying program and system parameters. Dynamic Load Balancing is exploited to obtain the most efficient computation, programming, and the routing method which corresponds with the varying states of network. Many recent authors, for example: [1], [3], analyzed and solved that problem, but most of them just mentioned a general framework which didn't implement to specific network; another limit is the increased amount of computations (as the cost also increases) to re-arrange the routing path among the processors. Obviously, the cost of pre- programming routing in Static Load Balancing is lower than the cost of adaptive routing in Dynamic Load Balancing. This is further suggested if we can combine the guaranteed Static Routing Path with Dynamic Load Balancing in assignment and distribution of workload among processors, that scheme is optimal to construct a scalable architecture. Indeed, this combination can probably implement?

To validate that solution, our study presents a new concept of Global Centralized Dynamic Load Balancing which is a product due to combination of Static Routing Path and Dynamic Load Balancing, that concept is not almost investigated.

The rest of paper is organized as follow. The next section looks at the approach of some researches about the properties and the Shortest Routing in HD* graph, and any definitions of Dynamic Load Balancing. In section 3, some studies of Dynamic Load Balancing and their related strategies. The proposed algorithm and framework are specified in section 4. Section 5 contains the discussion and our conclusion.

2 Background

In this section we look at some properties and the Shortest Routing in Hyper-De Bruijn-Aster [8]. Dynamic Load Balancing is also mentioned in literature.

2.1 Hyper-De Bruijn-Aster Graph (HD*)

Firstly, we present some regular methods of Routing Path in Hypercube Network and Routing Path in De Bruijn Network. Then we describe Shortest Routing in HD*.

2.1.1 Routing Path in Binary Hypercube Network (BHB Network)

In BHB Network, the Routing Path is conventional, the bit differences between the source and destination of a path are resolved from low dimension to high.

Two nodes in BHB are connected by an edge if the Hamming distance between the two nodes is 1 i.e., the number of positions where the bits differ in the binary labels of two nodes is 1. For Routing Path in BHB Network, there are many ways. One of the most regular Routing Path in BHB is that we choose the edge corresponding to a left-shift of current label and the last bit being the next bit of the destination node.

For example, Routing Path in figure 1 when packets are transferred from source node (0000) to destination node (1111) as follows:

- (0000)→ (0001) → (0011) → (0111) → (1111).

An edge between any two vertices if and only if their node labels differ in exactly one bit. Consequently, if the label of source node and the label of destination node are different M bits

which then generates (M-1) Intermediate nodes. For example above, the source node (0000) and the destination node (1111) are different 4 bits, thereby achieving (4-1)= 3 Intermediate nodes. We assume that in BHB, the computation of amount Intermediate nodes that the packets (are transferred by given source node) pass by and the path selection process for those packets can pre-program, this is capable Static Oriented Routing in BHB.

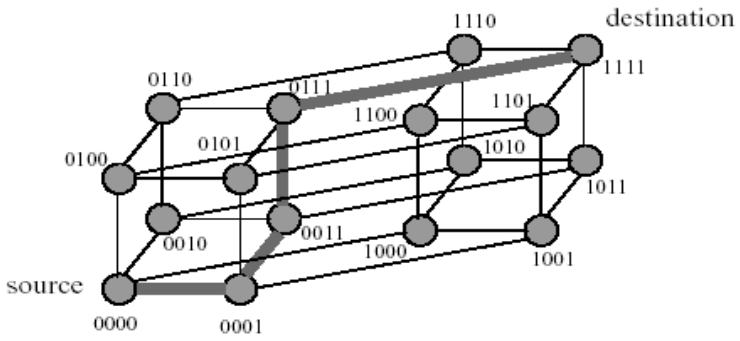

Figure1: An example of Hypercube(4)

2.1.2 Routing Path in Binary De Bruin Network (BDB Network)

In BDB Network, the greedy routing path is simply and optimal. The Shortest Routing algorithm in BDB is analyzed and improved valuably in [3], [7]. In this paper, we just would like to present a basic algorithm of Routing in BDB Network.

Routing method is as follows: Let $V = v_n v_{n-1} \ldots v_1$ is the source node and $W = w_n w_{n-1} \ldots w_1$ is the destination node. The routing path is $v_n v_{n-1} \ldots v_1 \rightarrow v_{n-1} \ldots v_1 w_n \rightarrow v_{n-2} \ldots v_1 w_n w_{n-1} \rightarrow \ldots \rightarrow w_n w_{n-1} \ldots w_1$. For example, let $V = 1001$ and $W = 1111$ the routing path is $1001 \rightarrow 0011 \rightarrow 0111 \rightarrow 1111$ The idea is that we choose the edge corresponding to a left-shift of current label and the last bit being the next bit of the destination node.

If we define the address of source node and destination node, relying on the rule of Routing Path in BDB Network, we can absolutely predict how to transmit the packets from source node to destination node, compute the amount of Intermediate nodes (these node are between source node and destination node that package is transmitted), suppose equally N nodes in which the package transferred, and their N labeling address. This is Static Oriented Routing in Network that means we can pre-program the Routing Path among Multiprocessors Network. Ability of Static Oriented Routing is also mentioned in HD*.

2.1.3 Routing Path in HD*

Routing algorithm (peer-to-peer) bases on Shortest Routing in HD*(m,d,n) [8] is little simply. In [8], we assume that the Routing Path between two arbitrary nodes v(h,d) and v(h',d') is established by using Shortest Routing as:

+ v(h,d) → v(h',d): only apply *Shortest Routing* in the Hypercube-part-label graph.
+ v(h',d) → v(h',d'): just refer to *Shortest Routing* in the De Bruijn-part-label graph.

Or we can invert this scheme in order to get the same result.

In the following the concept above, we get a formula:

Shortest Routing(HD) = Shortest Routing(H) + Shortest Routing(D)*

In case of we invert Shortest Routing of Hypercube and De Bruin in that formula, the Shortest Routing in HD* is unchanged.

2.2 Shortest Routing in Binary Hyper-De Bruijn-Aster BHD*(m,2,n)

With the characteristics of Shortest Routing in HD* above, we then consider the Shortest Routing in BHD*(m,2,n) with {d=2}.

Number of nodes in BHD* (m,2,n) is: $2^m x\ 2^n$.

Following the Shortest Routing in HD*, we propose the "firing rule" of Shortest Routing strategy in BHD* as::

Shortest Routing in (BHD) = Shortest Routing in (BH) + Shortest Routing in (BD)*

Routing Path of the packet is the Path traversed by the packet from its source node A (H_n, D_n) to its destination node B (H_d, D_d) in BHD* follow basic steps below:

Step 1: We just consider Shortest Routing related to the Binary Hypercube-part-label of node A and B ($H_n \rightarrow H_d$) and we do not refer to Binary De Bruijn-part-label. Basing on Shortest Routing in BHB (see 2.1.1), Routing Path then traverses (M-1) Intermediate nodes.

Step 2: We just consider Shortest Routing related to the Binary De Bruijn-part-label of node A and B ($D_n \rightarrow D_d$)and we do not refer to Binary Hypercube-part-label. Relying on Shortest Routing in BD (see 2.1.2), Routing Path then traverses (M-1) Intermediate nodes.

Step3: We only consider Routing Path established as: $D_d \rightarrow H_n$. Therefore, number of Intermediate traversed node is 1 node.

For the independence of Routing Path between BHB and BD in BHD*, the total of Intermediate nodes in which the packet traversed in BHD* is:

(M-1+N+1)=M+N= K node.

Both M nodes and N nodes have defined labels (or defined address). According to that remark, K nodes in BHD* also contain defined node labels.

The Shortest Routing of HD* are unchanged with varying (d), thereby these considerations of BHD*(m,2,n) are also suitable for model HD*(m,d,n) {with extended (d)}.

We have already proposed the concepts of Shortest Routing (in case of source node and destination node are given) which are predictable; this makes the pre-programming of Static Oriented Routing in Global Dynamic Load Balancing become simpler than when we set Routing Path among processors at the variable moments . In the following, we can easily compute and predict the cost of communication; the memory of CPU is therefore not expensive.

2.3 Dynamic Load Balancing Strategies

According to ([1] M. Zaki), there are four different strategies differing along two axes. The techniques are either *global* or *local*, answer the question of what information will be used to make a load balancing decision, and they are either *centralized* or *distributed*, both of which define where load balancing decision are made. In a centralized scheme, the load balancer is located on one master processor node and all decisions are made there. In a distributed scheme, the load balancer is replicated on all processors.

Global Strategies: In the global scheme, the load balancer uses the performance profiles of all available processors. The global schemes are subdivided into two techniques.

- *Global Centralized DLB (GCDLB):* In this scheme the load balancer is located on a master processor (centralized).After calculating the new distribution, and profitability of work movement, the load balancer sends instructions to the processors who have to send work to others, indicating the recipient and the amount of work to be moved. The receiving processors just wait till they have collected the amount of work they need.
- *Global Distributed DLB (GDDLB):* In this scheme the load balancer is replicated on all the processors. In GDDLB, the profile information is broadcast to every other processor. This also eliminates the need for the load balancer to send out instructions, as that information is available to all the processors. The receiving processors wait for work, while the sending processors transfer the data

Local Strategies: In local scheme processors are partitioned into different groups of size K. The partitioning is usually done such that each group has nearly equal aggregate computational power, or by considering the physical proximity of machines. There are also two different local strategies below.

- *Local Centralized DLB (LCDLB):* In this scheme, there is one centralized Load Balancer, which asynchronously handles all the different groups. Once it receives the profile information from one group, it sends instructions for re-distribution for that group before proceeding to the other groups.
- *Local Distributed DLB (LDDLB):* In this scheme, the load balancer is replicated on all the processors, but the profile information is broadcast only to the members of the group.

2.4 Comparisons between Global vs. Local Strategies, Distributed vs. Centralized Strategies

Global vs. Local Strategies

The choice of a global or local scheme depends on the behavior an application will exhibit. The benefit in a local scheme is that performance profile information is only exchanged within the group. For global schemes, balanced load convergence is faster compared to a local scheme since all processors are considered at the same time. However, this requires additional communication and synchronization between the various processors; the local scheme minimizes this extra overhead. But the reduced synchronization between processors is also a downfall of the local scheme if the various group exhibit major differences in performance. For example, if one group has processors with poor performance (high load), and another group has very fast processors (little or no load), the later will finish quite early while the former group is overload.

Distributed vs. Centralized Strategies

There are tradeoffs associated with choosing one location scheme over the other. For centralized schemes, the reliance on one central point of balancing control could limit future scalability. Additionally, the central scheme also requires an “all-to-one” exchange of distribution instructions from the balancer to the processors. The distributed scheme helps solve the scalability problems, but at the expense of an “all-to-all” broadcast of profile information

between processors. However, the distributed scheme avoids the "one-to-all" distribution exchanges since the distribution decisions are made on each processor.

3 Some related studies of Dynamic Load Balancing

In this section, we look at some related studies of Dynamic Load Balancing strategies and our judges concern with those studies.

3.1 Dynamic scheduling

Predicting the Future: A common approach of load balancing in Multiprocessors Network or Network of Workstations is to predict future performance due to past information. For example, in [9], a global distributed scheme is presented, and load balancing involves periodic information exchanges.V. Saletore [11] implements a local distributed receiver-initiated scheme; M.Zaki [1] exploits an interrupt–based receiver-initiated scheme.

Task Queue Model: A host of approaches have been proposed in the literature targeting shared memory machines. These fall under *the task queue model*, where there is a logically central task queue of loop iterations. Once the processors have finished their assigned portion, more work is obtained from this queue.

Single Program Multiple Data Computation Model: The Single Program Multiple Data (SPMD) paradigm implies that all the processors run the same code, but operate on different sets of data. The motivation for using SPMD programs is that they can be designed and implemented easily, and they can be applied to a wide range of applications such as numerical optimization problems and solving coupled partial differential equations (Malik [3])

Most of studies above concentrate on constructing the general framework Dynamic Load Balancing, respectively. The lack of those issues is not mentioned about implement Dynamic Load Balancing in specific Multiprocessors Network. They are maybe applicable to the general applications, but if we place them into the different graphs, are they still effective? For example, if we apply *Predict Future* algorithm in Star graph, Mesh graph, or De Bruijn, will we get the same optimal result? Obviously, each graph has different features from others, if we implement the same algorithm to them, we will evidently take the different results.

M.Zaki[1], provides a detail method for Dynamic Load Balancing in Network of Workstations in four steps: 1) Monitoring processors performance, 2) Exchanging this information between processors, 3) Calculating new distributions and making the work movement decision, 4) The actual data movement.

The basic steps above, the goal of both step 1 and step 2 is Synchronization. In their approach, synchronization is triggered by the first processor that finishes its portion of the work. This processor the sends an interrupt to all the other actives processors, who send their performance profiles to the load balancer.

Two aspects of this Synchronization are not clearly: How to define which processor finishes its portion of the work before the other processors? thereby the all of processors necessitate have an index " time-threshold" which is the time based to complete theirs works. Relying on that time based, all processors will compare their finished time of processing work with "time-threshold", then the first processor is found. The problem is how to get the "time-threshold"? The second problem, all processors are capable to send the interrupt to others, if the number of processors increases, it takes a high cost to exploit.

In order to avoid these limits above, our strategy focus on matching the extremely efficient HD* with a specific Global Dynamic Load Balancing. That optimal combination is expressed by combining Static Oriented Routing (or Shortest Routing) in HD* with our proposed Global Dynamic Centralized Load Balancing.

4 Global Centralized Dynamic Load Balancing combined with Shortest Routing in BHD*

HD* (with arbitrary degree) and BHD* have the same characteristics of Shortest Routing [8]. In this paper we just address Shortest Routing in BHD* combined with Global Dynamic Load Balancing in order to make the computation easier , but this work does not change the features of Shortest Routing in HD* .

In this section, we present our model of Global Centralized Dynamic Load Balancing, and define some modeling parameters that may influence the performance of our scheme. Then we present our proposed Algorithm of Global Centralized Dynamic Load Balancing combined with Shortest Routing in BHD*.

4.1 Model of Global Centralized Dynamic Load Balancing

The goal of Global Centralized Dynamic Load Balancing Strategy bases on setting the Load Balancer (Master processor) to be a defined node in BHD*. Therefore, the Routing Path between Load Balancer and arbitrary node is predicted and established easily.

All processors are placed and arranged with Binary Hyper-De Bruijn-Aster Graph BHD*(m,2,n).

The number of processors (or nodes) is: $2^m x 2^n$ = P nodes.

In initial condition, nodes in BHD* are designed that they consist of homogeneous pro-cessors which have the similar properties: the processor speed, the memory size, and the available disk space. The processor speeds of all processors are alike and unchanging du-ring of run- time processing. The communications between nodes are always optimal (all processors communicate to each other with high signaling speed). Therefore, the cost of communication among processors is skipped over.

BHD* consists of (P-1) processors (slaves); a node Load Balancer (Master pro-cessor) that is supposed node P. Load Balancer is responsible for:

1) Middle machine communicates the outside networks with inside BHD*.
2) Receiving input data, and subdivides data into the small tasks.
2) Monitoring performance of processors.
3) Calculating the new distribution and making the work movement decision.
4) Sending the actual data (tasks) to the processors.

All processors have the same functions and duties:

1) Communicating with their neighbors.
2) Sending their profile information to Load Balancer.
3) Receiving tasks from Load Balancer, processing them and returning the results to Load Balancer.

4.2 Modeling Parameters and Definitions

Processor parameters: 1) Fixed number of processors available for the computation, is denoted as (P-1) nodes, 2) Processor speeds of all processors are alike, denoted as V_0.
Load Unit (lu): Unit of data (size of L was selected depending on memory size and processor speed), stored at processors and wait for be executed by processors.
Migrating Unit (mu): The input data (from outside Networks) was sent to Load Balancer; Load Balancer stores it and partitions it into small tasks. Then migrate them to processors, each processor can execute W times of Migrating Unit (W_{mu}).
Workload: *Weight* of each node, can process W times of Load Unit (W_{lu}), denoted as the current capacity of load at each node. The Workload of each node varies at updating moment of system. At initial time, all processors have the same maximum value of Workload, denoted as W_0.

The total *weight* by the migrating units assign to each processor should be proportional to the performance of the processor, in order to balance the workload.
Proposition: *Size of Load Unit (lu) = Size of Migrating Unit (mu) =L*
Generalization: *Weight of Migrating Units assigning to each processor* (W_{mu}) = *Weight of Load Units* (W_{lu}) *that each processor is capable to execute them*.
The time of computation necessitates finishing the assigned work at each processor:

$$T_{comp} = W_i / V_O$$

The cost of computation C_{comp} at each node is proportional with T_{comp}(the higher T_{comp}, the higher the cost C_{comp}).
The cost of performing communication (at each processor {i}) with Load Balancer and the cost communication (from Load Balancer to processor {i}):

$$C_{comm} = \text{Amount of intermediate nodes} = K_i = \text{constant}$$

4.3 Proposed algorithm for Global Dynamic Load Balancing Model

The goals of algorithm contain two schemes: The first scheme is the Routing Path strategy between nodes vs. Load Balancer of which relies on technique of *Table-driven Routing*. The second scheme is our proposed algorithm for Global Centralized Dynamic Load Balancing in BHD*.

4.3.1 Exchange data and communicate between Nodes and Load Balancer

One of the most popular techniques for communication between processors and Master processor is *Table-driven Routing*. By applying that technique, the Routing control can pre-program and arrange self-scheduling.

To exchange data and communicate with Load Balancer, the basic required construction of each arbitrary processor (n) includes: 1) Storing and memorizing the address of Intermediate nodes (described by binary address) when a packet is transferred from that node (n) to Load Balancer, 2) Updating the physical link with closely neighbors.

Load Balancer is given a construction scheme as: 1) Storing and memorizing address of each node n (n=1÷{P-1}), and address of Intermediate nodes when packet is transferred from Load Balancer to node (n), 2) Updating the physical link with closely neighbors.

The duplex communication between nodes and Load Balancer is mainly based on looking for the destination node address (in terms of the source node address is given) collected by *Table-driven Routing* of each node. In the following, the *Table-driven Routing* at each node only updates the output address (namely address of closely neighbors) having available physical link.

A packet is set a *Heade*r before transferring. A *Header* contains : Address of source node, address of destination node, and address of Intermediate nodes.

Content of a *Header* of packets which are transferred from node (n) to Load Balancer:

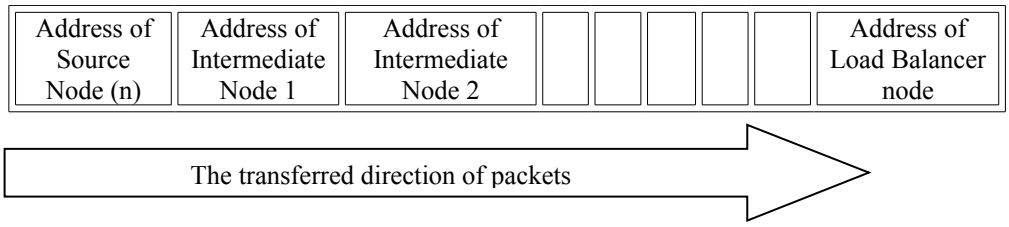

Figure 2: Structure of Header of Packets transferred from Node (n) to Load Balancer.

Communication and exchangeable data between node (n) and Load Balancer are done in basic steps below: Initially, the transferred packets of node (n) is set to add a *Header,* node (n) relies on its *Table-driven Routing* to make a physical link with the closely neighbor who has similar address with address of the Intermediate node 1 and then node (n) transfers packets to it. When packets arrive Intermediate node 1 in which continues doing the similar work, finding out the address of closely neighbor that has the same address of Intermediate node 2. In the following, the packets continue being transmitted until they arrive Load Balancer node (P).

Content of a *Header* of packets which are transferred from Load Balancer to arbitrary node :

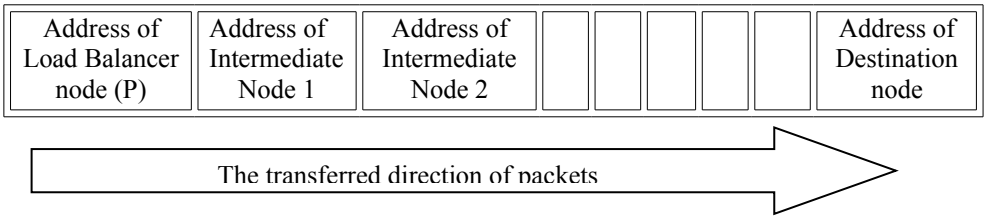

Figure 3: Structure of Header of Packets transferred from Load Balancer to arbitrary node.

We assume that the path traversed by the packets from Load Balancer node (P) to arbitrary node is done in basic steps above.

4.3.2 Proposed Algorithm of Global Centralized Dynamic Load Balancing

BHD* Network contains the processors possessing the maximum capacity of workload W_0.

The total of available resource of BHD is: (P-1)* $\times W_0$.

Firstly, Load Balancer receives the input data (from outside Networks); the strategy of Global Centralized Dynamic Load Balancing is started with the *initial step*: Load Balancer divides the input data into (P-1) parts, each part consists of W_0 (mu). Then Load Balancer sends (P-1) parts to each processor.

After the initial assignment of work to each processor, Global Centralized Dynamic Load Balancing is done in three steps:

1) Monitoring performance of BHD* Network
2) Calculating new distributions and making the work movement decision
3) Actual data movement.

Synchronization: In our approach, synchronization is triggered when Load Balancer receives the result from the first processor node (i) that finishes its portion of the work. Load Balancer then sends a request to all processors, who then send their performance profiles (their capacities of workload W) to Load Balancer.

Performance Metric: After receiving the performance profile of each processor, Load Balancer calculates *the total of available resource of BHD** :

$$\textit{The total of available resource of } BHD^* = W_0(node\,i) + \sum_{j=1, j\neq i}^{P-1} W_j$$

The totals of migrated works belong to *the total of available resource of BHD** . In the following, Load Balancer partitions the input data into (P-1) parts (part 1{W_1 }, part 2 {W_2 },.., part i {W_0 },.., part (P-1) {W_{P-1} }), each part contains size of data which is equal with W_n(mu) of each processor (n) {for n= 1÷(P-1)}.

Data Movement: (P-1) parts are also (P-1) works migrated to (P-1) processors after.

Data executed at each processor: After receiving works proportional to its capacity of workload W(mu), each processor then executes work. Each processor necessitates spending (T_i) $_{comp}$ so that it can finishes executing its work.

Results returned to Load Balancer: After finishing their works, all processors return the results to Load Balancer.

4.4 Modeling − Total Cost Derivation

We now present the cost model for our proposed Global Centralized Dynamic Load Balancing which is considered since Load Balancer receives the first result, re-distributes works until Load Balancer gets the returned results from all processor. The cost of a scheme is computed approximately, can be broken into the following categories as: cost of synchronization, cost of calculating new distribution, cost of data movement and cost of results returned to Load Balancer.

Cost of Synchronization

The synchronization involves the sending of requests from Load Balancer to (P-1) processors, who then send their performance profiles to the Load Balancer.

The cost of Synchronization *(C_{syn})* includes the cost of sending requests from Load Balancer to all processor *(C_{comm} {one-to-all})* and the cost of returning the responses of all processors to Load Balancer *(C_{comm} {all-to-one})*.

In the following, this cost is specified in terms of the kind of communication required for synchronization . The cost is given below:

- C_{comm} {one-to-all} = Totals of Intermediate nodes = $\sum_{i=1}^{P-1} K_i$ (nodes)
- C_{comm} {all-to-one} = Totals of Intermediate nodes = $\sum_{i=1}^{P-1} K_i$ (nodes)
- $C_{syn} = C_{comm}$ {one-to-all} + C_{comm} {all-to-one} = $2 \times \sum_{i=1}^{P-1} K_i$ (nodes) (1)

Cost of Distribution Calculation

We assume that Load Balancer has the rapid advances in high speed computation power and memory size. Therefore, this cost is usually small, and we denoted it as C_{Dist}. (2)

Cost of Data Movement

We compute *Cost of Data Movement* approximately. That cost also is considered in terms of the totals of Intermediate nodes that data (sent from Load Balancer to (P-1) processors) forwarded.

C_{mov} {one-to-all} = Totals of Intermediate nodes = $\sum_{i=1}^{P-1} K_i$ (nodes). (3)

Cost of execution of data

Each processor finishes executing the assigned work with $(T_i)_{comp}$. Cost of execution of data is considered equally that includes totals of the necessary time to finish executing work W_i *by processor speed* V_O.

.

$$T_{comp} = \sum_{i=1}^{P-1} (T_i)_{comp} \quad (4)$$

Cost of returning results to Load Balancer

This cost is similar to cost of returning response to Load Balancer. Therefore, it is given below.

C_{return} {all-to-one} = $\sum_{i=1}^{P-1} K_i$ (nodes). (5)

Total Cost

The above set of recurrence relations can be solved to obtain the cost of each step. The total cost is approximately measured by classified to synthesize five equations {(1),(2),(3),(4),(5)}, thereby getting the total cost of the Global strategies as

$$TC = C_{syn} + C_{Dist} + C_{mov} + T_{comp} + C_{return} = 2 \times \sum_{i=1}^{P-1} K_i + C_{Dist} + \sum_{i=1}^{P-1} K_i + \sum_{i=1}^{P-1} (T_i)_{comp} + \sum_{i=1}^{P-1} K_i$$

Or

$$TC = 4 \times \sum_{i=1}^{P-1} K_i + C_{Dist} + \sum_{i=1}^{P-1} (T_i)_{comp}$$

Where *(P-1)* is total processors in BHD*, K_i is the defined number of Intermediate nodes that packets (are transferred from source node (i) to destination node) pass by, and C_{Dist} is the cost of Distribution Calculation when Load Balancer computes the new re-distribution. $(T_i)_{comp}$ is the necessary time that processor(i) finishes executing assigned work W_i with processor speed V_O.

5 Discussion

Assignment of work to each processor proportional to its performance workload *W* and profiting the Shortest Routing in BHD*, our model provides a solution to adapt with the dynamic changes in Network. Obviously, our achieved results are approximately. But in fact, as if we exploit this model, the performance of our Network systems will be more increasing than we predicted. In contrast to our comment, the remark is present as.

The performance of Workload W_i of each processor is achieved at the moment (t_1) when Load Balancer sends request to that processor is a predictable value which is different from the workload considered at the moment of receiving assigned work (t_2). From (t_1) to (t_2), processor executed more works that generated the higher performance of Workload. We assume if the assigned work is unchanged, the increased performance of Workload allows the execution time of that work decreasing. In the following, at (t_2) the performance of Workload of Network increases, that makes evidently the T_{comp} decrease. The decrease of T_{comp} is equally to reduce the *Cost of Execution of Data.* Consequently, the *Total Cost* is also reduced.

We now look at the open problem of our model: What is the solution for poor performance when the Network HD* has many failure nodes?

As mentioned in the preceding section 2.2, Shortest Routing in HD* can provide a lot of Routing Paths; every Routing Paths has the same number of Intermediate nodes. By adding more ability Intermediate nodes to Header of each processor, we achieve more possibly Routing Paths. Though there are many failure nodes, the communication between all processors is not congested or stopped.

We suppose that Load Balancer is failure, our Network also becomes failed. In this case, we propose that we set many replicated Load Balancer as a group. We consider that group is only a node labeled by one address. When a Load Balancer is failure, the other Load Balancer will continue executing works until the failed Load Balancer is fixed.

6 Conclusion

In this paper, we proposed the new approach, Global Centralized Dynamic Load Balancing combining with Shortest Routing in HD*. We also present an efficient trade-off between Static Routing and Dynamic Load Balancing. We have shown that the computation of five kinds of cost being simpler, can also predicted. By distributing assigned work proportional to its performance Workload *W* , our strategy tries to keep the processors as busy as possible and avoiding situations in which processor sits idle waiting for work to be done.

Moreover, our strategy becomes a best candidate for the real networks with high scalability and complexity. Our proposed algorithm can also efficiently be exploited in Local Centralized Dynamic Load Balancing. We have observed that our algorithm quickly achieves a stable situation with a fixed number of processors.

There are some ways in which this work can be extended. Our model can apply not only in Homogeneous Networks, but also in Heterogeneous Networks, for example: in NOWS (Network of Workstations), or in VLSI interconnection architecture.

References

[1] M. Zaki, W. Li, and S. Parthasarathy. Customized dynamic load balancing for a NOW. In *5th IEEE Intl. Symp.High-Performance Distributed Computing, also TR 602, Univ. of Rochester, Dec 1995.*

[2] Robert Elsässer, Burkhard Monien, and Stefan Schamberger. Load Balancing in Dynamic Networks. DELIS-TR-OO46.Project Number 001907.

[3] S*hahzad Malik. (219762)* Dynamic Load Balancing in a Network of Work-stations 95.515F Research Report November 29, 2000 *Rochester*, Aug. 1996.

[4] Friedhelm Meyer auf der Heide and Berthold Vöcking. Shortest Path Routing in Arbitrary Networks. Journal of Algorithms, Vol 31 (1), 1999.

[5] Wei Shi and Pradip K Srimani, "*Hyper-Butterfly Network: A Scalable Optimally Fault Tolerant Architecture"*.

[6].Jyh-Wen Mao and Chang-Biau Yang, "*Shortest path routing and fault tolerant routing on de Bruijn networks*", Journal of Networks,Vol. 35,Issue 3,Pages 207-215 2000.

[7]. Ngoc Chi Nguyen, Nhat Minh Dinh Vo and Sungyoung Lee, "Fault Tolerant routing and broadcasting in the de Bruijn Networks", the 19th IEEE International Conference on Advanced Information Networking and Applications (AINA2005), Taiwan, Mar. 28-30, 2005.

[8] Nguyen Hong Thai, Pham Minh Tri, Nguyen Chi Ngoc: "Evolved Fault Tolerant Routing, Broadcasting and Load Balancing in Hyper-DeBruijn-Aster Network"

[9] N. Nedeljkovic and M. J. Quinn. Data-parallel programming on a network of heterogeneous workstations.*1st HPDC*, Sep 1992.

[10] Gary Shao, Rich Wolski and Fran Berman. Performance Effects of Scheduling Strategies for Master/Slave Distributed Applications. UCSD CSE Dept. Technical Report # CS98-598, September 1998

[11] V. Saletore et al. Parallel computations on the charm heterogeneous workstation cluster. *3rd HPDC*, Apr 1994.

Biographies

Pham Minh Tri is a student in Telecommunication Department at Ho Chi Minh City University of Technology, South Vietnam. His research interests include Static and Dynamic Load Balancing, Applications of Graph Models in Multiprocessors Network and Parallel and Distributed Systems.

Nguyen Hong Thai is a student in Telecommunication Department at Ho Chi Minh City University of Technology, South Vietnam. His research interests contain Optimize Fault Tolerant Routing, Broadcasting and Load Balancing in Network Models, and Applications supported by Open Sources.

Nguyen Chi Ngoc is a Master of Computer Engineering (KyungHee University). He is a professor in Telecommunication Department at Ho Chi Minh City University of Technology, South Vietnam. He received the Best Research Student Award of Kyung Hee University. His Interested Fields conclude Fault Tolerance and Security for Wireless Communication, Steganography and Information Hiding, Fault Tolerance and Security for Ubiquitous Computing and Routing on Parallel Computing.