Analysis and Investigation of Nearest Neighbor Algorithm for Load Balancing

Javed Hussain¹, Durgesh Kumar Mishra²

¹Research Scholar, Mewar University, Chittorgarh (Raj.), India and Computer Centre, Vikram University, Ujjain (M.P.), India.
²Department of Computer Science and Engineering, Shri Aurobindo Institute of Technology, Indore (M.P.), India.

Abstract: In this paper, we focus on nearest-neighbor load-balancing algorithms where each processor only considers its immediate neighbor processors to perform load balancing actions. Nearest neighbor load-balancing algorithms have emerged as one of the most important techniques for parallel computing based on direct networks. Load balancing through nearest neighbor algorithm processes make decisions with information of local work load and distribute it load within its neighborhood. Here we highlight performance of different topologies of a network in simulated environment.

Keywords: algorithm, Chain, 2D, 3D Mesh network, nearest neighbor

1. INTRODUCTION
Advances in hardware and software technologies have increased interest in the use of large-scale parallel and distributed systems for database, real-time, and large-scale scientific and commercial applications. The operating system and management of the concurrent processes constitute integral parts of the parallel and distributed environments. One of the biggest issues in such a system is the development of effective techniques for the distribution of processes among processing elements to achieve some performance goal(s), such as minimizing execution time, minimizing communication delays, and/or maximizing resource utilization. Load-balancing is one of the most important problems which have to be solved in order to enable the efficient use of multiprocessor systems. Load-balancing aims at improving the performance of multiprocessor systems by equalizing the computational load over all processors in the system since it is commonly agreed that equally balancing loads between all processors in the system directly leads to a minimization of total execution time. Load-balancing is performed by transferring load from heavily to lightly loaded processors. For that purpose, a load-balancing algorithm has to resolve the issues of when to invoke a balancing operation, whom makes load-balancing decisions according to what information and how to manage load migrations between processors. We can find several answers to these questions which result in a wide set of load-balancing techniques. A highly popular class of load-balancing strategies is nearest-neighbor approaches which are edge-local, that is, methods that can be implemented in a local manner by each processor consulting only its neighbors, thereby avoiding expensive global communication in distributed applications. The load moved along each edge is related to the gradient in the loads across it. These kinds of distributed load-balancing algorithms are appealingly simple and they degrade gracefully in the presence of asynchrony and faults. Most of these algorithms are implemented in an iterative way to achieve a global load balanced state and, therefore, they are referred to as iterative load-balancing algorithms.

2. RELATED LITERATURE
With nearest neighbor algorithm each processor considers only its immediate neighbor processors to perform load balancing operations. A processor takes the balancing decision depending on the load it has and the load information to its immediate neighbors. By exchanging the load successively to the neighboring nodes the system attains a global balanced load state. The nearest neighbor algorithm is mainly divided into two categories which are diffusion method and dimension exchange method. With this method a heavily or lightly loaded processor balances its load simultaneously with all its nearest neighbors at a time while in dimension exchange method a processor balances its load successively with its neighbor one at a time [7]. Nearest neighbor load balancing algorithms rely on successive approximation to a global uniform distribution, and hence at each operation, need only be concerned with the direction of workload migration and the issue of how to apportion excess workloads. There are a number of ways for the choice of the direction of workload migration. Among them, we are interested in a couple of simple representatives, the diffusion (DF, for short) and the dimension exchange (DE for short) methods. With diffusion method, a highly or lightly loaded processor balances its workload with all of its nearest neighbors simultaneously in a load balancing operation [9]. With the dimension exchange method, by contrast, a processor in need of load balancing balances its workload successively with its neighbors one at a time and its new workload index will be considered in the subsequent pairwise balancing [9,10,11]. They are closely related because they lend themselves particularly well to implementation in two basic communication architectures, the all-port and the one-port models, respectively. The port model allows a processor to
exchange messages with all its direct neighbors simultaneously in a communication step, while the one-port model restricts processor to exchange messages with at most one direct neighbor at a time. Both of these two models are valid in real parallel computers and were assumed in many recent researches on communication algorithms ([15], for example). In an asynchronous implementation of load balancing, processors perform load balancing operations discretely based on their own local workload distributions and invocation policies. Since load balancing algorithms can be treated as orthogonal to their invocation policies, we consider the load balancing operations of processors in one time step so as to isolate their effects on the system imbalance factor from the effects of invocation policies. We focus on the static situation load balancing in which the underlying computation in a processor is suspended while the processor is performing load balancing operations. The dynamic situation makes only a few differences to the analysis of the effects of load balancing. [16], we made a comparison between two classes of nearest neighbor load balancing algorithms, the dimension exchange (DE) and the diffusion (DF) methods, with respect to their efficiency in driving any initial workload distribution to a uniform distribution and their ability in controlling the growth of variance among processors’ workloads. We focused on their four instances the ADE, the ODE, the ADF and the ODF which are most common versions in practice. The comparison was made comprehensively in both one-port and all-port communication models with consideration of various implementation strategies: synchronous/asynchronous invocation policies and static/dynamic random workload behaviors. [10]. We showed that the DE method outperforms the DF method in the one-port communication model. In particular, the ODE algorithm is best suited for synchronous implementation in the static situation. We also revealed of the superiority of the DE method in synchronous load balancing even in all-port communication model. The strength of the diffusion method is in asynchronous implementation in the all-port communication model. The ODF algorithm performs best in high dimensional networks in that case. The comparative study not only provides an insight into nearest neighbor load balancing algorithms, but also over practical guidelines to system developers in designing load balancing architectures for various parallel computational paradigms.

3. THEORETICAL DESCRIPTION
The nearest-neighbor-averaging (NNA) is completely local load balancing method[1]. The idea is to change the load of each processor such that it is equal to the mean load of the processor and its neighbors.

Notation:
- $\alpha$: diffusion parameter, $N$: number of processor, $i$: processor number, $i = 1, ..., N$, $R$: vector of current load processors, $R = [L_1, L_2, ..., L_N]$, $L_i$: current load of processor $i$, $\Delta(i)$: a set of directly neighbouring processors, $\delta_{ij}$: a load transferred from processor $i$ to $j$.

In every step, a load $\delta_{ij}$ is transferred from processor $i$ to every neighbor $j \in \Delta(i)$.

After one load balancing step, at time $t+1$, the processor $i$ should have a load $L_i(t+1)$.

$$L_i(t+1) = \frac{1}{1+\alpha} \left( L_i(t) + \sum_{j \in \Delta(i)} L_j(t) \right)$$

The Nearest Neighbor Averaging is an asynchronous variant of NNA. In this variant, when a processor is highly loaded then it transfers a portion of its load to all deficient neighbors. The amount of transferred load is proportional to the difference of the mean load and the load of the neighbor. Let the deficiency of each neighboring processor $j \in \Delta(i)$ for the processor $i$ be given by $h_j = \max(0, L_i(t+1) - L_j(t))$ and the total deficiency by $H_i = \sum_{j \in \Delta(i)} h_j$. Then, the asynchronous NNA performs a load transfer $S_i(j)$ from processor $i$ to each of its neighbors $\delta_{ij}\Delta(i)$ with

$$S_i(j) = (L_i(t) - L_j(t+1)) \frac{h_j}{H_i}$$

4. RESULT
Assumption:
Above approaches have strict locality of the communication and the control. Assuming that tasks are indivisible, identical and task numbers are integers. Due to the assumption load balancing process can be simulate for many tasks. Initial Load on at every Processor is fixed.

A. Topology = Chain

<table>
<thead>
<tr>
<th>Processors</th>
<th>Time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>16</td>
<td>13</td>
</tr>
<tr>
<td>24</td>
<td>14</td>
</tr>
<tr>
<td>36</td>
<td>15</td>
</tr>
<tr>
<td>48</td>
<td>17</td>
</tr>
</tbody>
</table>

![Figure 1](image)

B. Topology = 2D Mesh

<table>
<thead>
<tr>
<th>Processors</th>
<th>Time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>
Fig. 2

TABLE III

<table>
<thead>
<tr>
<th>Processors</th>
<th>Time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>24</td>
<td>5</td>
</tr>
<tr>
<td>36</td>
<td>8</td>
</tr>
<tr>
<td>48</td>
<td>10</td>
</tr>
</tbody>
</table>

Fig. 3

Analysis
Fig. 1 shows for chain topology, here analyzing the stability of the network we find that as number of processor increases up to 12, stability time of chain topology increases very fast but greater than 12 number of processor, the stability time increases slowly. Fig. 2 shows 2D Mesh topology in which time for stability is less as compared to chain topology, it goes smoothly till 12 number of processors but from 12 to 24 number of processors stability time increases faster, after that as we increases the number of processors the stability time increases slowly. Fig. 3 shows 3D Mesh topology in which graph goes almost smoothly and takes minimum time to become stable.

5. CONCLUSION
Comparing all three graphs as shown in fig. 1, 2 and 3 resulting from simulation result we conclude that 3D Mesh network perform better compare to Chain and 2D Mesh network because this network take lesser time for stability and from Fig. 3 we can analyze that the transfer of load to neighbor processors are almost uniform and similar. Therefore it is better than chain and 2D networks.

References
[1] Dominik Henrich, “The Liquid Model Load Balancing Method” journal of parallel algorithm and applications, special issue on algorithms for enhanced mesh architectures,