Chapter 9

Proposed A New Distributed Load Balancing Algorithm


Chapter 9: Proposed A New Distributed Load Balancing Algorithm

9.1 Introduction

A distributed computer system is a collection of processors connected through a network that works together for a common purpose. The primary objective of a distributed system is to proper utilization of the available resources in distributed environment. The most crucial resource is CPU speed and bandwidth of the underlying network. The low bandwidth may bottleneck the CPU speed. So, most commonly used mechanism is to share the load among the nodes by transferring some of the load from a heavily loaded processor to a lightly loaded processor. The load balancing improves the performance of the system by using the processing power of the entire system more effectively [154].

The distribution of loads to the processing elements is simply called the load balancing problem. In a system with multiple nodes there is a very high chance that some nodes will be idle while the other will be over loaded. The goal of the load balancing algorithms is to maintain the load to each processing element such that all the processing elements become neither overloaded nor idle that means each processing element ideally has equal load at any moment of time during execution to obtain the maximum performance (minimum execution time) of the system [51, 79, 155, 156, 157]. So the proper design of a load balancing algorithm may significantly improve the performance of the system.

In the network there will be some fast computing nodes and slow computing nodes. If we do not account the processing speed and communication speed (bandwidth), the performance of the overall system will be restricted by the slowest running node in the network [51]. Thus load balancing strategies balance the loads across the nodes by preventing the nodes to be idle and the other nodes to be overwhelmed. Furthermore, load balancing strategies removes the idleness of any node at run time.

Load balancing is the way of distributing load units (jobs or tasks) across a set of processors which are connected to a network which may be distributed across the globe. The excess load or remaining unexecuted load from a processor is migrated to other processors which have load below the threshold load [158]. Threshold load is such an amount of load to a processor that decides whether any load may come further to that processor or to send. In a system with multiple nodes there is a very high chance that
some nodes will be idle while the other will be over loaded. So the processors in a system can be identified according to their present load as heavily loaded processors (more jobs waiting than threshold load), lightly loaded processors (less jobs waiting than threshold load) and idle processors (virtually have no job to execute). By load balancing strategy it is possible to make every processor equally busy and to finish the works approximately at the same time.

There are two fundamental approaches to the load balancing algorithm design. In static load balancing design approach the tasks are assigned on the basis of a priori knowledge of a system and once the tasks are allocated on the nodes do not change [3]. The performance of the static load balancing algorithms depends on the prior information about the tasks and the system. The decision to transfer the tasks does not depend on the system state change. So this approach is best suited for homogeneous distributed computing system. But the dynamic load balancing algorithms take the decision to transfer the tasks depending on the current state of the system. The tasks are transferred from heavily loaded node to the lightly loaded node [3, 71, 159]. So the quality of dynamic load balancing algorithms depends on the collection of information on load on different nodes in the system. So this approach is best suited for heterogeneous distributed computing system.

In dynamic load balancing the information may be collected either by centralized or distributed approach. In centralized approach the information is collected by a specially designed central node and in distributed approach each node has the autonomy to collect the information about the load of the system. It has been reported that the collection of information by centralized approach about the system state does not cause any performance degradation for a reasonably large distributed computing systems [160]. The drawback of this approach is that the performance of a globally distributed system would be very poor and the cost of state information collection would be too much and maintaining a huge information by a single node will surely cause a performance degradation. In the distributed information collection policy the information is collected either by sender initiative or receiver initiative algorithm. In sender initiative approach the heavily loaded nodes search for lightly loaded nodes for transferring extra load and the receiver initiative approach is the converse of sender initiated approach. In this
approach either a sender or a receiver may poll all the nodes in a network for load balancing causing huge overheads. To reduce the overheads the sender or receiver nodes poll a selected number of nodes like nearest neighbors [13,20,99,116]. Another performance problem with this approach is associated with the inter-arrival times and service times.

9.2 Computational Model and Assumptions

The distributed system is represented by an undirected graph $G = (P, E)$ where $P$ refers to the set of processors and $E$ to the set of links [163]. The communication link between any two processors is assumed to be bidirectional. Thus if there is a link $(P_i, P_j)$ that joins $P_i$ with $P_j$, then $P_i$ is a neighbor of $P_j$ and they can send and receive information and load from each other. We also assumed that there are $N$ heterogeneous processors $P_n$ where $n=2, 3, \ldots, N$.

Each processor maintains a local load table that holds the three field of information: processor ID, status and load of neighboring nodes. Status of a node is either -1, 0 or +1. 0 implies for normal loaded, +1 for overloaded and -1 for lightly loaded situation. A node will be lightly loaded if it's ready queue becomes less than half the ready queue length. From load table a node will select a least loaded node. If still load becomes excess, then select next least loaded node and so on.

A processor $P_n$ maintains two queues for its tasks: a 'ready' queue and a 'waiting' queue as shown in Fig. 9.1. A task or a job enter into the system as input to a processor directly either from outside or as a transfer from a neighbor. Thus, a processor has either local tasks or remote tasks to be executed. The task, which is directly entered to a processor queue, is known as local task and on the other hand, the task, which is transferred from the other neighboring processors, is known as remote task. The 'ready' queue contains a buffer of finite size of length six and all tasks in the ready queue is executed by the respective processors. The 'waiting' queue has a buffer of infinite size and holds the external incoming tasks to it. If the 'ready' queue is not full, waiting queued tasks is transferred to the 'ready' queue otherwise task is sent for remote processing. A processor having half filled 'ready' queue is considered as lightly loaded while overloaded nodes are
those whose 'ready' queue is filled and its 'waiting' queue is not vacant. Lightly loaded
nodes always receive the incoming tasks in their ready queues. Normally loaded nodes
have 'ready' queue full and the 'waiting' queue empty. The system performance is
evaluated in terms of its overall response time along with the service time, the queuing
time and the transmission time.

![Simulation Model](image)

**Figure 9.1** Simulation Model

### 9.3 Simulation Study

We considered a mesh topology of sixteen heterogeneous nodes for the simulation
purpose. Each queue follows the M/M/1 queue model. Once a task is assigned in ready
queue cannot be migrated. Only the tasks from the waiting queue are allowed to be
migrated. We also considered a node as under loaded if it has load below the half of the
length of a ready queue; a node is considered to be moderate loaded if its waiting queue
is empty but ready is not empty and a node is over loaded if its waiting queue is not
empty. Whenever a node either becomes under loaded or over loaded, it informs its status
to its neighbors and the neighboring nodes would immediately update their load tables. A
over loaded node will compare its amount of load with the loads from its load table with
status. Once the over loaded node finds the under loaded node having highest load
deficiency, it will set an agreement with that under loaded node that it wants to transfer
the load. If the under loaded node agrees then load would be transferred. The over loaded
node may transfer its load to more than one nodes if its load is more than the load
deficiency of a single node. Fig. 9.2 represents a situation of node P6 and its neighbor P2, P5, P7 and P10. Each node represents two numbers: first number shows the status and second number shows the load. Fig.9.3 represents the load table at P6 for that moment. Suppose that P6 has total tasks 12, and then it has extra 6 tasks which are to be transferred. 4 out of 6 is transferred to P5 and rest 2 tasks is transferred to P2 node.

![Heterogeneous Mesh Topology of Node 16](image)

**Figure 9.2** Heterogeneous Mesh Topology of Node 16

<table>
<thead>
<tr>
<th>Node</th>
<th>Status</th>
<th>Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2</td>
<td>-1</td>
<td>2</td>
</tr>
<tr>
<td>P5</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>P7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P10</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

**Figure 9.3** Load Table

We measured the response time with the arrival rate as shown by Fig.9.4 and number of load message with arrival rate shown by 9.5 and we compared our algorithm with the algorithm without load balancing and Ni's drafting algorithm [42, 159, 164].
Chapter 9: Proposed A New Distributed Load Balancing Algorithm

Figure 9.4 Average Response Time VS Arrival Rate

Figure 9.5 Number of Load Message VS Arrival Rate
Chapter 9: Proposed A New Distributed Load Balancing Algorithm

9.4 Summary

In this chapter a new distributed load balancing algorithm has been presented. In this algorithm each node maintains a load table holding the current load situation of its neighbors. A new concept in threshold load in each node avoids unexpected over loading and also ensures more job allocation towards more powerful nodes. As the nodes inform their load situation on status change, the over loaded or under loaded nodes do not need to poll for transfer of jobs or to invite the jobs from the neighbors and thus causes low congestion in the network. Response time and overheads are measured by applying this algorithm in mesh topology of sixteen nodes and compared the new algorithm with Ni’s drafting algorithm. The new algorithm produces better response time and over heads with respect to the arrival rate than Ni’s drafting algorithm. In our algorithm, the response time is better than Ni’s drafting algorithm by 12.1% and over heads is better than Ni’s drafting algorithm by 10.3% with respect to the arrival rate.