# A Table Driven Dynamic Load Balancing Scheme for Distributed System

**Antim Panghal**[1], **A.K Sharma**[2]

[1]M. Tech student B.S.A.I.T.M Faridabad, Haryana, India

[2]P.G Dean of Research B.S.A.I.T.M Faridabad, Haryana, India

**Abstract:** ***In a distributed network of different computing hosts, the performance of the distributed system depends on how the work is efficiently divided among all the participating nodes. Load balancing is an important activity wherein tasks are assigned to participating nodes with a view to minimize the waiting time and bring improvement in execution time of the system. The existing load balancing techniques consider each participating node having equal computing capability .In this work we propose a load distribution algorithm for a cluster of heterogeneous commodity hardware. The algorithm considers the nature of applications at hand before allocating the computing resources. A considerable improvement in execution time of parallel applications has been observed especially when tasks are sent and processed on favorable nodes instead of randomly assigned nodes.***

**Keywords:** Dynamic load balancing, Distributed system, Distributed Algorithm, data-intensive task, computation intensive task.

## 1. Introduction

A distributed System is a collection of nodes that cooperate and coordinate with each other to achieve a common goal of efficient utilization of computing power. Therefore a CPU becomes the most important resource which is shared between the participating nodes of the distributed system. One of the important mechanisms for utilizing and sharing the CPUs optimally is the policy of balancing the load amongst the nodes. This type of load balancing can be achieved by transferring some of the tasks from a heavily loaded node to lightly loaded node . The possibility of transferring a task arises either when the processing time for a task in a processor is expected to be larger than that of another remote processor, or when the imbalance of workload at various processors is sufficiently larger. The load balancing improves the performance of the system by using the processing power of the entire system more effectively.

Load balancing algorithms may either be static or dynamic, depending upon the rules they follow. In static load balancing algorithms, on the basis of the time needed to complete any task, tasks are assigned to processors during the compile time and their relation is determined. No decision is taken regarding shifting of a task from one processor to another during the execution time. But in dynamic load balancing algorithms load status at any given moment is used for taking us to which task is to be shifted from one processor to another. Load balancing algorithms mainly depend upon four components (1) transfer policy (2) location policy (3) selection policy (4) information policy. In this work location policy is being constructed. According to location policy decentralized algorithm is categorized into sender-initiated algorithm, receiver-initiated algorithm and symmetrically-initiated algorithm [4].

Emerging application in the field of Science, Engineering and commerce require intensive computations as well as storage and analysis of huge amount of data. Since last few years, primary attention was to maximize CPU cycle per second which result in exponential increase in clock speed of uni-processor CPUs. Multicore processor architecture further improved computing performance by adding more than one core as an independent processing element on same die. In addition to computation-intensive requirements, today's scientific and commercial application need performing computation over large datasets typically ranging from tens to hundreds of gigabytes or terabytes. Moreover these applications require dynamic scalable solution for data-intensive computing. In this paper, we propose algorithm for solving data-intensive and computation-intensive problems based on individual capability of nodes.

## 2. Related Work

Various approaches for balancing the load has been described in the literature, Ni et al.[11] have described a mechanism for a homogeneous distributed system where every node maintains a load table containing the states of neighboring nodes. In [5] two algorithms based on Source and Destination have been compared .The source based algorithm called "Sender-Initiated Algorithm" identifies an over-loaded sender nodes which take the initiative to request the under-loaded nodes (receiver) to receive the job. The destination based algorithm called "Receiver-Initiate Algorithm" identifies the under-loaded receiver nodes which take the initiative to invite tightly loaded nodes to send their job. The result suggests that if the system is lightly loaded, the sender –initiated policy is better as compared to receiver-initiated policy. The receiver-initiative policy is better in case of heavily loaded system. Similar conclusion has been drawn by Y. Wang and R.J.T. Morrisin [13].

Another algorithm described in the literature determines the optimal load of a Heterogeneous distributed system [12].This algorithm is also static in the sense that the decision regarding transfer of task does not depend on the system state. If required, the node may transfer some of its task to a selected node. However a task thus transferred will cause a communication delay as well as queuing delay at the node so selected.

Krueger and Finkel [1] have suggested an algorithm called “Above-Average Algorithm”. The transfer policy of this algorithm broadcasts messages to achieve consensus about average load in the network across machines that are over and under-loaded. There by algorithm maintaining the load at each node within an acceptable range of the system average. Its location policy has following two components: (1) Sender-Initiated components, in which a sender node broadcast a Too High message and set a Too High timeout alarm. Thereafter listen for an accept message from other nodes until the time out expire (2) Receiver-Initiated component, in which a node ,on becoming a receiver ,broadcast a Too Low message ,set a Too Low timeout alarm. Thereafter start listing for a Too High message.

Dynamic load balancing algorithms developed by a number of authors considered each participating nodes as having equal computing capability [15][16]. A critical look at the available literature indicates that the above discussed algorithms, in general broadcast the large number of messages for load distribution across the distributed system and considered each participating nodes as having equal computing capability.

In this paper, we propose a load distributing algorithm wherein load is distributed among the participating nodes that depends upon the nature of the application whether it is computation-intensive or data intensive and analyzes the computing power of individual nodes which will improve the execution performance of the system.

## 3. Proposed Work

The proposed method divides the task at hand into two categories, computation-intensive and data intensive [17]. The computation intensive problem requires more CPU cycles than memory usage whereas Data-intensive problem requires more memory usage and data transfer than CPU cycle. Therefore for efficient utilization of resources in network of workstations, depending upon the hardware attributes the participating nodes of the network are divided into two clusters: computation-intensive and data-intensive clusters of workstations. Consequently, the system distributes the load to the appropriate cluster depending upon the type of the tasks at hand being computation intensive or data intensive. In fact the load is distributed towards the best supporting group of workstation for efficient use of resources.

In this work, a teach node a table called load Tab is maintained as shown in table1. The table contains the information about the participating nodes in terms of their group type and current load.

**Table1**. Sample of two separate groups for different nature of problems

| *Node No.* | *Group Id.* | *Current load* |
|---|---|---|
| 1 | $G_{11}$ | 0 |
| 2 | $G_{21}$ | 1 |
| 3 | $G_{22}$ | 2 |
| 4 | $G_{12}$ | 2 |
| 5 | $G_{23}$ | 4 |
| 6 | $G_{13}$ | 3 |
| 7 | $G_{14}$ | 3 |
| 8 | $G_{24}$ | 5 |

Since a task (T) can be of two types. Let us define a tag called Load type to represent the type of the task i.e. ‘load type’ can assume values 0 and 1 for computation-intensive and data-intensive task respectively. The algorithm that distributes the tasks in the distributing environment is given below. In this algorithm, a variable called disk flag is being used that indicates whether a task at hand has been distributed or not depending upon the information returned by diskflagi.e. 0 or 1.

```
Algorithm distributeGroup(loadType,T)
{
int i=0,j=0, diskflag=0;
if(loadType(T)= = 0)
{ //computation –intensive problem
while((diskflag = = 0)&&(i<=m))
{
i++
diskflag= distload(T,G1i)// ith node of group G1
}
}
else
{// data-intensive problem
while((diskflag==0)&&(j<=n))
{
j++
diskflag= distload(T,G2j) // jth node of group G2
}
}
}
```

It may be noted that the above algorithm uses a function called distload(). A detailed discussion on the working of its algorithm is given below.

For the purpose of distribution of load the proposed mechanism computes the average load of a group of nodes and thereafter based on, following the location and transfer policies the task are distributed.

**Transfer policy:** The transfer policy [1] is a threshold policy that uses two adaptive thresholds called lower threshold and upper threshold. These thresholds are equidistant from the node’s estimate of the average load across all nodes. For example, if a node’s estimate of average load is 3, then the lower threshold=1 and upper threshold=5.The current work load is the load at a given time of a ready process in the system.

Now, a node whose current load is less than the lower threshold is considered as receiver, whereas the node whose current load is greater than the upper threshold is considered as sender. Nodes that contain the current loads between these threshold lie within the acceptable range, so they are considered neither sender nor receivers nodes.

**Location policy:** A node computes its current load information with the help of the load table, and classifies itself as a sender or a receiver. Thereafter the locations of other sender/receiver nodes are identified as per the procedure given below.

**(1) Sender node**

As soon as a task arrives, the sender node (a node whose current load greater than the upper threshold) searches for a possible receiver node in the network by consulting information contained in its load table. If a receiver is found then the sender node sends the task to the same. On receiving the task the receiver node broadcast a message to all participating nodes about its current load .Thereafter all nodes update their load table accordingly.

**(2) Receiver node**

As soon as a task arrives the Receiver node (a node whose current load is less than the lower threshold) receives the task. After receiving the task the receiver node broadcast a message to all participating nodes about its current load .Thereafter all nodes update their load table accordingly.

The algorithmic detail of the function distload() is given below:
Let a task T arrives at a node N. Let LT and UT represents lower and upper threshold values , RN and SN represents sender and receiver nodes , proc Ready is a queue in which all ready process are placed for execution.

```
Algorithmintdistload(T,N)
{
while(1)
{
if(load(N)<LT)
nodeType=RN
elseif(load(N)>UT)
nodeType=SN
else
return0
if(nodeType==RN)
{
placeT onprocReady queue.
broadcast update(message)
return1
 }
else
{
RN=findReceiver(loadtable)
distload(T,RN)
return1
 }
}
}
```

findReceiver(loadtable) // It finds a receiver node by consulting information contained in load table of the node in question.

Example: Let us computes the following terms for the data given in table 1.

1. **Average load** of G1=(0+2+3+3)/4=2
2. **Lower threshold** of G1=1
3. **Upper threshold** of G2=3
4. **Average load** of G2=(1+2+4+5)/4=3
5. **Lower threshold** of G2=1
6. **Upper threshold** of G2=5

The network of nodes consists of N(say 8) nodes wherein N/2(say4) nodes belong to group G1 and N/2(say4) nodes belong to G2. We know that in a distributed system as soon as a new task joins the system it can execute equally well on any node of its group type, independent of the node where it has joined. Suppose a new task T whose work load=1and loadtype =0arrives at node 1. Thereafter the node 1 consults its loadtaband finds the following:

1. The type of task is same as its own type.
2. It accepts the arrived task because its load is less than the upper threshold.
3. The current load of node 1 is within the range of system average.
4. The current load of node 1 after receiving the new task is l, which is also within acceptable range.
5. The task is placed in the ready queue of node1. Accordingly ,the loadtab is updated as shown in table2.

**Table 2**

| Node No. | Group Id. | Current load |
|---|---|---|
| 1 | $G_{11}$ | 1 |
| 2 | $G_{21}$ | 1 |
| 3 | $G_{22}$ | 2 |
| 4 | $G_{12}$ | 2 |
| 5 | $G_{23}$ | 4 |
| 6 | $G_{13}$ | 3 |
| 7 | $G_{14}$ | 3 |
| 8 | $G_{24}$ | 5 |

Similarly let us consider a new task whose current load=2and loadtype=1 arrives at node 3. Thereafter the node 3 consults its loadtab and finds the following:
1. The type of task is same as its own type.
2. It accepts the arrived task because its load is less than the upperthreshold.
3. The current load of node 3 is within the range of system average.
4. The current load of node 3 after receiving the new task is 4, which is also within acceptable range.
5. The task is placed in the ready queue of node3. Accordingly the loadtab is updated as shown in table3.

**Table 3**

| Node No. | Group Id. | Current load |
|---|---|---|
| 1 | $G_{11}$ | 1 |
| 2 | $G_{21}$ | 1 |
| 3 | $G_{22}$ | 4 |
| 4 | $G_{12}$ | 2 |
| 5 | $G_{23}$ | 4 |
| 6 | $G_{13}$ | 3 |
| 7 | $G_{14}$ | 3 |
| 8 | $G_{24}$ | 5 |

Kindly note that each time a task arrives the existing system broadcasts messages to gather the information about various nodes of the network for possible distribution of load. This reoccurring wastage of time of broadcasting the message has been saved by the proposed method. It uses an eager strategy to collect the information of load of various nodes and store it into a load table called 'loadtab'. Thereafter the table is updated only when the load conditions of a node change which is a very less frequent activity. Hence forth a node searches the potential sender or receiver nodes merely by consulting its load table instead of broadcasting messages. However the distribution of task within a group further improves the execution time.

## 4. Conclusion

In this work a group specific dynamic load balancing method has been proposed that maintains a load table at each node. The potential nodes for load distribution are identified by consulting its load table and not by broadcasting messages .Thereby reducing the network congestion. The distribution of task with in a group further improves the execution time.

## References

[1] Krueger,P., and R.Finkel, "Adaptive Load Balancing Algorithm for a Multicomputer", Technical Report 539,University of Wisconsin-Madison,Apr.1984
[2] Urjashree Patil, and Raja Shree Shedge, "Improve Hybrid Dynamic Load Balancing Algorithm For Distributed Environment", International Journal of Scientific and Research Publication ,Volume 3,issuse3,March 2013.
[3] Ankush P. Deshmukh and Prof. Kumar swamy Pamu, "Applying Load Balancing: A Dynamic Approach", International Journal of Advance Research in Computer Science and Software Engineering ",Volume 2,issue 6,June 2012.
[4] PrakashKumar,Pradeep Kumar and VikasKumar,"An Effective Dynamic Load Balancing Algorithm for Grid System", International Journal of Engineering and Technology ,Volume 4,issues 8,August 2013.
[5] D.L.Eager et al .,"A comparision of receiver-initiated and sender-initiated adaptive load sharing", performance evaluation 6(1986) 53-68.
[6] Ali M.Alakeel,"A Guide to Dynamic Load Balancing in Distributed Computer System ", International Journal of Computer Science and Network Security, Volume 10.No. 6,june 2010
[7] ToufikTaibi, AbdelouahabAbid, Engku Fariez Engku Azahan ,"A Comparison of Dynamic load Balancing Algorithms", J.J Appl. Science, Volume 9, Issue 2, 2007.
[8] Eager,D.L.,D.Lazowska, and J.Zahorjan,"adaptive load sharing in homogeneous Dynamic load Balancing Algorithms", J.J Appl. Science, Volume 9, Issue 2, 2007.
[9] Shivaratri,N.G.,andP. Krueger, "Two Adaptive Location Policies for Global Scheduling ,"proceedings of the 10th international conference on distributed computing system,May 1990,pp.502-509
[10] Krueger,P.,and M.Livny,"The Diverse Objectives of Distributed Scheduling Policies,'proceeding of the 7th International Conference on Distributed Computing System,Sept.1987,pp.242-249.
[11] L.M. Ni et al.," A distributed drafting algorithm for load balancing", IEEE Trans. Software Enginrg. SE-11 (10) (1985) 1153-1161.
[12] A.N. Tantawi and D. Towsley," Optimal load balancing in distributed computing system", J. Assoc. Comput. Ma-chinery 32 (1985) 445-465.
[13] Y. Wang and R.J.T. Morris, "Load sharing in distributed system", IEEE Trans. Comput. C-34, (3) (1985) 204-217.
[14] Harshal Khandre ,Prof. Manasi Kulkarni, " A New Approach for Dynamic Load Balancing Algorithm", International Journal of
[15] Advance Research in Computer Engineering and Technology ,Volume 2,issues 6,june 2013.
[16] MaisNijim ,TaoXie ,Xiao Qin, "Performance Analysis of an Admission Controller for CPU and I/O Intensive Application in
[17] self Managing Computer System ," ACM Operating system review ,Vol .39 ,No.4,2005,
[18] Pushpender Chandra, Bibhudatta Sahoo, "A Novel Load Balancing Algorithm for I/O Intensive Load in Heterogeneous Cluster," International Conference on Advance in Computing, February 2008.
[19] Rajkumar Sharma, Dr. Priyesh Kanungo "Dynamic Load Balancing in Network of Workstations of Different Computing Power. "ph.D. Thesis, Devi ahilya vishwavidyalaya, Indore.
[20] HarshalKhandre ,Prof.ManasiKulkarni," A New Approach for Dynamic Load Balancing Algorithm", International Journal of Advance Research in Computer Engineering and Technology ,Volume 2,issues 6,june 2013