Load Balancing in Distributed Systems using Diffusion Technique

P. Neelakantan
Department of CSE, SVUCE, Tirupati, India

ABSTRACT

The purpose of load balancing algorithm is to distribute the excess load from heavily loaded nodes to underloaded nodes. A new dynamic load balancing algorithm is proposed based on diffusion approach (DDD) for homogeneous systems where the processing capacities of all nodes in the system are equal. The proposed algorithm works iteratively to balance the load among the nodes in a system. The dynamic distributed diffusion algorithm has been developed for coarse and large granularity applications, where the load shall be treated as an Integer quantity. The functioning of the proposed algorithm is demonstrated by using a random graph & simulation has shown the proposed algorithm performs better in terms of time taken to balance the load, minimizing the load variance among the nodes and maximizing the throughput.

Keywords

Load balancing, diffusion

1. INTRODUCTION

A well known and popular load balancing approach first introduced by Cybenko and Boillat [1][2] is diffusion load balancing. The algorithm works iteratively to balance the load among the nodes in a system. The idea behind this algorithm is in each round, the overloaded node exchanges its excess load with all neighbors individually. The advantage of diffusion algorithm lies in the collection of information from the nodes in a system. The information policies will have higher impact on earlier completion of load balancing algorithm.

The diffusion algorithm [3, 4, 5] collect information from a group of nodes which is referred to as a domain. The algorithm tries to balance the loads in each domain, such that excess load will be transferred to the under loaded node in their domain thus reducing communication overhead. When loads are to be transferred to the large radii node in a system it puts more communication overhead which forfeits the advantages of load balancing algorithm. The objective of the diffusion load balancing algorithm in both static load situations and in dynamic load situations is to keep the nodes to contain an equal number of loads. To do this, the loads are distributed evenly among the nodes as quickly as possible. Much work has been done under the assumption that every edge is only allowed to forward one load unit per round [6, 7, 8, 9] or a constant number of loads can be passed by each node [10, 11].

In a simple diffusion algorithm, if the neighboring nodes of any $node\ i$ have the load value smaller than underlying $node\ i$ then those neighbors are referred as underloaded loaded nodes. Once underloaded neighbours are determined, the underlying node will measure the load difference between itself and each one of its neighbours. Then, a fixed portion of the excess load is sent to each one of the under loaded neighbours. This strategy, as well as other strategies from the literature based on this, [5][12][13] is originally conceived under the assumption that load can be divided arbitrarily ,i.e., the load is treated as a non-negative real quantity. The load is treated as integer quantity in medium and large grain parallelism (back-track searches, branch-and-bound optimizations, theorem proving, interpretation of PROLOG programs, adaptive refinement techniques for solving PDEs, and ray tracing)which are more realistic assumption and common in practical parallel computing environments as carried out in [14] [15[11]]. A relevant strategy in this area is the SID (Sender Initiated Diffusion) algorithm [12]. Luling and Monien [13] devised a distributed load balancing algorithm for a grid with load index defined as a summation of service times of jobs currently running in a node but does not consider the effects of communication latency. Liu et al. [19] used an agent based system, which migrates the excess workload from the heavily loaded nodes to the lightly loaded nodes, and they assumed the nodes in the system are homogeneous. Acker et al., [20] proposed a new decentralized dynamic diffusion algorithm that is capable of dynamically adapting to changing operating parameters. The devised load balancing algorithm runs on every node is decentralized and dynamic. It handles the nodes in the systems that are heterogeneous in terms of node processing capacity, architecture and network speed.

2. NOTATIONS & ASSUMPTIONS

Consider a distributed system with N nodes represented as an undirected graph $G= \{V, E\}$, where $V = \{1,2, \dots N\}$ a set of nodes and E set of edges connecting the nodes. The set of nodes that have direct links with $node\ i$ is represented by $D_i = \{j | j \in V \text{ and } (i,j) \in E\}$ called neighbor nodes [6][8][11].

Notations:

N: Number of nodes in a distributed system

V: set of nodes in a distributed system N=|V|

$L_i(t)$: Load of node i at time t $\forall i$

D_i: Domain of node i at time t $=\{j | j \in V \text{ and } (i,j) \in E\}$

$\sum_{j \in D_i} L_j(t)$: Load of node j that belongs to domain of node i at time t

$\bar{L}_i(t)$: Average load of the domain of node i given by $\bar{L}_i(t) = \frac{L_i(t)+\sum_{j \in D_i} L_j(t)}{|D_i|+1}$

$\mu(t)$: Average System load $= \frac{1}{N}\sum_{i=1}^{N} L_i(t)$ at time t

$Var(t)$: Variance of system, $\sigma^2(t) = \frac{\sum_{i=1}^{N}(L_i(t)-\mu)^2}{N}$

$dev_{ij}(t)$: Load deficit at node j at time t , $\forall j \epsilon D_i$

$deviation_i(t)$: Excess load of node i at time t

$L_i^{max}(t)$: Maximum load of node in the domain of node i

$L_i^{min}(t)$: Minimum load of node in the domain of node i

$x_{ij}(t)$: Apportion of the load sent to the deficit neighbor j in the domain of node i

$\square_i(\mathbf{t})$: Fine load distribution by node i

It is assumed that a distributed system consists of identical high performance nodes (homogeneous system) connected by a set of high bandwidth communication links in order to provide a powerful computing platform to execute computationally intensive parallel and distributed applications [11].

It has been assumed a distributed system consists of independent parallel jobs [5] [6] [7] where they can be executed at any time, in any order and at any node. It is assumed that the jobs have varied service times and are modeled as a uniform distribution. The arrival of the jobs is a Poisson distribution.

3. MATHEMATICAL MODEL

Let $L_i(t)$ is the load value of node i in a system of N nodes at given time instant t and the load values among the nodes in the system are represented by using load vector $L(t) = (L_1(t) \dots L_N(t))$. The load values of a node can be real or non negative integer values depending on the granularity of the application. Let the load value of every node in a system is represented as an integer quantity. When the load is represented as an integer quantity the maximum load difference in the system between any two nodes is to be 0 or 1.

Let the load balancing algorithm is initiated at time t and it takes some time t_1 where $t_1 > t$ to balance the load among the nodes. Let the average system load at time t is given by

$$\bar{L} = \mu(t) = \frac{1}{N}\sum_{i=1}^{N} L_i(t) \qquad (3.1)$$

Since load balancing algorithms are load conservative, i.e., they do not neither create nor destroy load but only move it around the system such that load values of individual nodes changes due to load balancing actions .In static load situations, the value of μ does not change over time. Thus, it is given by

$$\mu(t) = \mu(0) = \mu \ \forall \ t$$

The imbalance of the system load at time t is measured with a synthetic indicator, the variance of the load of the nodes, i.e., their quadratic deviation from μ.

$$\sigma^2(t) = \frac{\sum_{i=1}^{N}(L_i(t)-\mu)^2}{N} \qquad (3.2)$$

If variance among the system is minimized, each node in the system contain equal loads to process so as to minimize the response time of the system.

Theorem:

The execution of a diffusive load balancing policy nullifies any load imbalance in a system[6][11], i.e.,

$$\exists t > 0 \Rightarrow \sigma^2(t) = \frac{\sum_{i=1}^{N}(L_i(t)-\mu)^2}{N} < 1 \ \ Where\ \mu = \frac{1}{N}\sum_{i=1}^{N} L_i(0) \qquad (3.3)$$

Lemma: If the variance among the nodes in a domain D is decreased by exchanging the loads only among the nodes in that domain by applying load balancing algorithm then the global variance of the system can also be decreased.

Proof:

Let us multiply the expression (3.2) by *N* and for the sake of simplicity exclude the right hand side value N which is multiplied with $\sigma^2(t)$ .The variance at time *t* can then be expressed as:

$$\sigma^2(t)=\sum_{i=1}^{N}(L_i(t)-\mu)^2 = {L_1}^2(t) + {L_2}^2(t) + \dots + {L_N}^2(t) - 2\mu\sum_{i=1}^{N} L_i(t) + N\mu^2 = \sum_{i=1}^{N} {L_i}^2(t) - N\mu^2 \qquad (3.4)$$

After initiating a load balancing activity at time *t* within a domain of *D* nodes, the variance at time $t+1$ can be expressed as:

$$\sigma^2(t+1)=\sum_{i=1}^{N}(L_i(t+1)-\mu)^2 = {L_1}^2(t+1) + {L_2}^2(t+1) + \dots + {L_N}^2(t+1) - 2\mu\sum_{i=1}^{N} L_i(t+1) + N\mu^2 = \sum_{i=1}^{N} {L_i}^2(t+1) - N\mu^2 \qquad (3.5)$$

The variation of the variance can be expressed as

$$\Delta\sigma^2(t+1,t) = \sigma^2(t+1) - \sigma^2(t) = \sum_{i=1}^{N} {L_i}^2(t+1) - \sum_{i=1}^{N} {L_i}^2(t) \qquad (3.6)$$

Let us consider the load balancing activity takes place in a domain D_i and not in the in other domains $\overline{D_i}$. The load balancing algorithm exchanges load in the nodes that are present in a domain D_i (by indicating$\overline{D_i}$, the set of nodes not belonging to the domain D_i and by $|D_i| = |V - \overline{D_i}|$the number of nodes involved in load balancing)

$$\mu_{D_i}(t) = \frac{1}{|D_i|}\sum_{i\in D_i} L_i(t), \qquad \mu_{D_i}(t+1) = \frac{1}{|D_i|}\sum_{i\in D_i} L_i(t+1)$$

$$\mu_{\overline{D_i}}(t) = \frac{1}{|V-D_i|}\sum_{i\notin D} L_i(t), \mu_{\overline{D_i}}(t+1) = \frac{1}{|V-D_i|}\sum_{i\notin D_i} L_i(t+1)$$

The equation (3.6) can also be rewritten as follows

$$\Delta\sigma^2(t+1,t) = \sigma^2(t+1) - \sigma^2(t) = \sum_{i\in D_i} {L_i}^2(t+1) + \sum_{i\notin D_i} {L_i}^2(t+1) - \left(\sum_{i\in D_i} {L_i}^2(t) + \sum_{i\notin D_i} {L_i}^2(t)\right) \qquad (3.7)$$

By adding and subtracting the terms $D_i{\mu_{D_i}}^2$, $\overline{D_i}{\mu_{\overline{D_i}}}^2$ in equation (3.7)

$$\Delta\sigma^2(t+1,t) = \sum_{i\in D_i} {L_i}^2(t+1) - D_i{\mu_{D_i}}^2 + \sum_{i\notin D} {L_i}^2(t+1) - \overline{D_i}{\mu_{\overline{D_i}}}^2 - \left(\left(\sum_{i\in D} {L_i}^2(t) - D_i{\mu_{D_i}}^2\right) + \left(\sum_{i\notin D} {L_i}^2(t) - \overline{D_i}{\mu_{\overline{D_i}}}^2\right)\right) \qquad (3.8)$$

The above equation can be expressed as

$$\Delta\sigma^2(t+1,t) = \Delta{\sigma_{D_i}}^2(t+1,t) + \Delta{\sigma_{\overline{D_i}}}^2(t+1,t) \qquad (3.9)$$

In other words, the variation of the global variance can be expressed as the variation of the variance in the sub systems identified by D_i and $\overline{D_i}$. The load of the nodes of $\overline{D_i}$ has not changed, i.e.,

$$\Delta{\sigma_{\overline{D_i}}}^2(t+1,t) = 0 \qquad (3.10)$$

Consequently, if the local action in D decreases the local variance

$$\Delta{\sigma_{D_i}}^2(t+1,t) < 1$$

a global benefit stems for the local load balancing action

$$\sigma^2(t+1) < \sigma^2(t)$$

4. PROPOSED METHOD

In a distributed system, the nodes exchange their load information at periodic intervals of time called information exchange interval T_s. The information exchange consists of load information of a node and the instant at which this information exchange takes place is called an information exchange epoch. In order to reduce the communication overhead, the information exchange is restricted only to the neighboring nodes.

Each node i receives a load information message from its neighbors which is kept in the node i memory. Due to communication delays induced by the network, each node i will have an estimate of the neighbor nodes load, because within the communication delay d_{ij} some load may be added to the node j or removed from the node j. The load information from node j to node i is represented $L_{ij}(t) = L_j(\tau_{ij}(t))$ where $\tau_{ij}(t)$ is a certain time instant satisfying$0 \leq \tau_{ij}(t) \leq t$. The node i, as d_{ii}=0 (delay is zero for the node i) will have exact information about its load.

A set of time instants is associated with each node for doing load balancing. At a given time instant, the node i executes the load balancing algorithm by comparing its load with the estimated load of its neighbors that are stored in the node i local memory during status exchange epoch. In order to analyze the DDD behavior, the variable t is discriminated by assuming the values t=0, 1, 2….

When the loads among the nodes are distributed randomly, a single iteration of the proposed load balancing algorithm consists of two procedures: procedure LB and procedure AccurateLB. In procedure LB, when the load of the node is greater than the average load of the domain of the node *i*, then that node is said to be an overloaded node, so it sends its excess load to the under load neighbors.

In the procedure AccurateLB of DDD, the node that initiated the load-balancing algorithm checks its underlying domain for balanced state. The domain attains balanced state if the load difference between two nodes in it is not greater than 1. If it is not balanced, a node that initiated the load balancing algorithm balances its domain in a refined way by sending messages to the overloaded nodes in its domain to distribute the loads to the under loaded neighbors

The node i compute the load average of its domain by taking the load information of the neighbors kept in the memory which is rounded to the nearest lowest integer value, which is given by

$$\bar{L}_i(t) = \frac{L_i(t)+\sum_{j\in D_i} L_j(t)}{|D_i|+1} \qquad (4.1)$$

After computing the load average, it evaluates the relative load weight to detect whether it is an overloaded node or under loaded node. For this purpose it uses the below formula

$$\text{deviation}_i(t) = L_i(t) - \bar{L}_i(t) \qquad (4.2)$$

From equation (2) If the value $deviation_i(t) > 0$ indicates node i is overloaded and it has to send its excess load to one of its deficient neighbors. The value $\text{deviation}_i(t) < 0$ indicates that the node is underloaded and there no need to transfer the load and hence no need to invoke the load balancing algorithm. Depending on the value of $deviation_i(t))$ the load balancing algorithm is initiated by the node i

4.1.1 *Load transfer calculation*

Once the node i determines that it is having an excess load, it has to distribute its excess load to the deficient neighbors. The node i form two sets Active_i and Send_idepending on the excess and deficit load values. Nodes having the deficit loads form the Active set which is denoted by Active_i and nodes having the excess loads form the set Send_i. After forming the two sets, for each deficit neighbor in set Active_i, load deficit for an individual node is stored which is given by

$$\text{Active}_i(t) = \{j \mid L_j(t) \leq \bar{L}_i(t)\} \text{ where } j \in D_i \cup \{i\}$$

$$\text{Send}_i(t) = \{k \mid L_j(t) > \bar{L}_i(t)\} \text{ where } k \in D_i \cup \{i\}$$

$$\text{dev}_{ij}(t) = \bar{L}_i(t) - L_j(t)$$

The total deficit of the domain of node i is calculated by using the formula:

$$TD_i(t) = \sum_{j\in\{\text{Active}_i\}} \text{dev}_{ij}(t) \qquad (4.3)$$

After determining the total deficit load, the node i proceed to determine how much portion of its excess load is to be sent for each deficit neighbor by having the below formula

$$x_{ij}(t) = \frac{\text{dev}_{ij}(t)}{TD_i(t)}(\text{deviation}_i(t)) \qquad (4.4)$$

4.1.2 Accurate Load Movements

The procedure AccurateLB is used by node i to check its domain for accurate balance. To do this, some additional parameters are required to probe for unbalanced domains. These parameters will do accurate load movements to balance the domain and hence to decrease the variance of the domain. The additional parameters introduced in the algorithm are:

A node that contains a maximum load value in the domain of : $Max\{Send_i\}$

A node that contains a minimum load value in the domain of i: $Min\{Active_i\}$

To detect the imbalance in domain of$node\ i$, the above two parameters will be used .

4.1.3 Load adjustments between nodes

After detecting, the imbalance in the domain of $node\ i$, some units of load must be moved between the nodes to reduce the variance in the domain of $node\ i$. To do this the following steps must be accomplished.

Step 1: In general excess loads are kept in $Send_i$ and deficit loads are kept in the set $Active_i$. When the load difference between the maximum loaded node in $Send_i$ and minimum loaded node in $Active_i$ is greater than 1 & all nodes in Active_i contains equal loads load refinement must be done such that all nodes in the domain must consist of nearly an equal amount of load.

It shall be easily known that if the load difference in the domain i.e., the load of the maximum loaded node in $Send_i$ when compared with the load of the minimum loaded node in

$Active_i$ contains load units greater than 1, some $\varphi_i(t)$ load units must be transferred. While transferring $\varphi_i(t)$ two constrains must be satisfied.

Constraint 1: The node in the set $Send_i$ after sending its excess load units to one of the deficient neighbors in the set $Active_i$,load value of the sender node must be equal or greater than one unit to the next higher load node in $Send_i$ is compared to avoid job shuttle between the nodes. From this, it will be concluded that the largest node in $Send_i$ remains as largest after sending $\varphi_i(t)$ units of load to least loaded deficit node in $Active_i$

Constraint 2: After receiving $\varphi_i(t)$ load units from the maximum loaded node in $Send_i$, the load value of the least loaded node in $Active_i$ must be equal to the load value of the next least loaded node in $Active_i$. The above two constraints plays a key role in avoiding job thrashing effect.

To analyze the time complexity of DDD algorithm, The single iteration of DDD in a node i involves distinct operations depending on the phase or phase of the algorithm to be executed to distribute the load among the neighbors. The below pseudo code of the algorithm shows the organization of different blocks and table 1 shows the maximum number of operations required at each level(The number of neighboring nodes to node i is denoted as d). By complete analysis, it shall be concluded that the time complexity of DDD is prevailed by the complexity of AccurateLB *O(d).*Hence it can be seen that the overall time complexity of the DDD is low when compared to the load balancing algorithms discussed in the literature survey.

Algorithm DDD

At each node i=1, 2…N

Compute $\bar{L}_i(t) = \left\lfloor \frac{L_i(t)+\sum_{j\in D_i} L_j(t)}{|D_i|+1} \right\rfloor$;

Find $Send_i = \{k | L_k(t) > \bar{L}_i(t)\}$ where $k \in D_i \cup \{i\}$

Find $Active_i = \{j | L_j(t) \leq \bar{L}_i(t)\}$ where $j \in D_i \cup \{i\}$

Compute $deviation_i(t) = L_i(t) - \bar{L}_i(t)$;

If ($deviation_i(t) > 0$) call **procedure LB**;

Choose an index $k \in Max\{send_i\}$ & $a \in Min\{Active_i\}$;

$\mathbf{L_i^{max}(t)} = L_k^{(t)}$;

$\mathbf{L_i^{min}(t)} = L_a^{(t)}$;

While ($\mathbf{L_i^{max}(t) - L_i^{min}(t) > 1}$)

Compute $\varphi_i(t) = \mathbf{L_i^{max}(t) - L_i^{min}} - 1$;

If ($\varphi_i(t) \leq 0$) exit;

Else

Call procedure AccurateLB;

Choose an index $k \in Max\{send_i\}$ & $a \in Min\{Active_i\}$;

$\mathbf{L_i^{max}(t)} = L_k^{(t)}$;

$\mathbf{L_i^{min}(t)} = L_a^{(t)}$;

End while

End DDD

Procedure LB

For each $j \in Active_i$

Compute $dev_{ij}(t)$;

End For

Compute $TD_i(t) = \sum dev_{ij}(t)$;

For each $j \in Active_i$

Compute $x_{ij}(t) = \frac{dev_{ij}(t)}{TD_i(t)} (deviation_i(t))$;

Transfer $\lfloor x_{ij}(t) \rfloor$ units to node j from node i;

End For

End LB

Procedure AccurateLB

If (i≠ *k*)

Node i sends message to k to transfer one load unit to *a* and the load at ***k*** is reduced by one unit.

Else

Node i transfer one load unit to *a* and the load at ***i*** is reduced by one unit.

End AccurateLB

DDD OPERATIONS			
	Actions	Operation	quantity
Initial actions	Check load Estimates of Neighbors	Memory access	d, the number of neighbors
	Evaluate average load	Addition Division	d 1
	deviation	Subtraction	1
	Active set, Send set	Comparison	d
Procedure LB	Evaluate deficit dev_{ij}	Subtraction	d
	Evaluate TD	Addition	d
	Evaluate x_{ij}	Multiplication Division	d
Procedure Accurate LB	Max element in the Send set and minimum element in the active set	Comparisons	O(d)
	Evaluate φ	Subtractions	d
	Send one message	Transmission	1

Table 1: Possible operations performed by DDD

4.2 CONVERGENCE OF THE PROPOSED ALGORITHM

To demonstrate DDD convergence, partially asynchronous assumption introduced by authors in [11][6][12]. The asynchronous algorithms are divided into two classes: totally asynchronous and partially asynchronous. Total asynchronous algorithms can tolerate large communication and computation delays but partially asynchronous algorithms require an upper bound on communication and computation delays to work correctly. This upper bound is denoted by U, which is applied to realistic load model assumed by DDD. The formal description of this assumption is given below. The time complexity of the proposed algorithm is O(d).

Assumption 1: Let us denote an upper bound U such that

a) For each node $i \in V$ and for every $t \in \mathbb{N}$, $\{t, t+1, \ldots t+U-1\} \cap T_i \neq \emptyset$

b) For each node $i \in V$, for every $t \in \mathbb{N}$ and for every $j \in D_i, t-U < \tau_{ij}(t) \leq t$

c) Node *i* sends the load to node *j* at time t, will be received by the node *j* before time t+U

d) Node *i*sends the instruction message to node *j* at time t, is received by the node *j* before time t+U

Part (a) of this assumption asserts that the load balancing step is performed by each node during time interval of length U. (b) states that load information of neighboring nodes kept by any node in a given time t are obtained at any time between t-U and t; (c) & (d) asserts that the instruction messages and load messages sent by node i to node j will not be delayed more than U time units.

To Prove DDD converge under assumption 1, two definitions and two lemmas are to be introduced.

Definition 1: $\text{Load}_1(t) = \min\{L_i(t') \mid i \in V, t-3U < t' \leq t\}$ is the minimum load value in a system during the time interval t-3U to t.

Definition2: $\text{Load}_k(t) = \min\{L_i(t') \mid i \in V,\ t-3U < t' \leq t, L_i(t') > \text{Load}_{k-1}(t), k > 1\}$, where $\text{Load}_k(t)$ is the minimum load value that occupies k-st place if the loads are sorted in non decreasing sequence during the time interval t-3U to t.

Lemma 1: The sequence of loads $(\text{Load}_1(t))$ $t \geq 0$, the time is increasing and upper bounded. There exist a time t1 such that

I. $\text{Load}_1(t) = \text{Load}_1(t_1)\ \forall\ t \geq t_1$

II. $r_{ji}(t) = x_{ij}(t) = \varphi_i(t) = 0\ \forall\ j, i \in V, \forall\ t \geq t_1$
$L_i(t_1) = \text{Load}_1(t_1)$

From Lemma1 it has been observed that at time t_1, the load value of all nodes become stable and no node would send or receive the load from its neighbors by executing the DDD algorithm.

Proof: Let a node i ∈ V and a time t ∈ ℕ. It is to be proven that $L_i(t+1) \geq \text{Load}_1(t)$.

If $t \notin T_i$ then $node\ i$ is not having the excess load, so it does not trigger the load balancing algorithm, instead it receives the load from the overloaded neighbors as a part of load balancing process which is initiated by some overloaded neighbor which is present in its domain. Thus

$$L_i(t+1) = L_i(t) + \sum_{j=1}^{N} r_{ji}(t) \geq L_i(t) \geq \text{Load}_1(t)$$

If $t \in T_i$, the node i has invoked the load balancing algorithm, where two different cases shall be found depending on which procedure in DDD is invoked in making the accurate load movements between the nodes.

Case 1: When there are no load movements has been generated by **AccurateLB(**$\varphi_i(t) = 0$**),** then there exits two different situations.

Situation 1: Node is an under loaded node ($\text{deviation}_i(t) \leq 0$). Then it will not send any load to its neighbors such that $x_{ij} = 0$ so $L_i(t+1) \geq L_i(t) \geq \text{Load}_1(t)$.

Situation 2: Node i is an overloaded node ($\text{deviation}_i(t) > 0$) . Then it will execute the procedure LB to migrate some load units to the deficient neighbors which is given by

$$L_i(t+1) = L_i(t) - \sum_{j=1}^{N} x_{ij} - \varphi_i(t) + \sum_{j=1}^{N} r_{ji} = \bar{L}_i(t) + \sum_{j=1}^{N} r_{ji} \geq \bar{L}_i(t).$$

Here $\bar{L}_i(t)$is the average load of the domain between the time interval t- 3U to t, so it is clear that $\bar{L}_i(t)$ is greater than $\text{Load}_1(t)$. Hence $L_i(t+1) \geq \text{Load}_1(t)$

Case 2: By applying the procedure AccurateLB, some load units move among the nodes ($\varphi_i(t) > 0$). It may be possible that an overloaded node does not transfer any load by using the **procedure LB**, but still there is chance to transfer some load units to the deficient neighbors in a more refined way to reduce the load variance in the system by calling the procedure AccurateLB. This happens in two different situations.

When the difference between the loads of first nodes in Active_i and Send_i is greater than 1, i.e., $\varphi = (L_i^{max}(t) - L_i^{min}(t) - 1), where\ \varphi > 1$ and the node i is having the maximum load in its domain then $\varphi = L_i(t) - L_i^{min}(t)$ is sent to the least loaded neighbors.

Thus $L_i(t+1) = L_i(t) - \varphi_i(t) + \sum_{j=1}^{N} r_{ji} \geq L_i(t) - \varphi_i(t) > L_i^{min}(t) \geq \text{Load}_1(t)$

When the difference between the loads of first node in Active_i and Send_i is greater than 1, i.e., $\varphi = (L_i^{max}(t) - L_i^{min}(t) - 1), where\ \varphi > 1$, node i is not having the maximum load in its domain but belongs to send_i , but there exists some node k with maximum load in domain of node i such that it sends an instruction message to k to send $\varphi = L_k^{max}(t) - L_i^{min}(t)$ is sent to the least loaded neighbors such that at some time $\tau \leq t$, the difference between $L_k(\tau) - L_i(\tau) \leq 1$ where $k, i \in send_i$. So, it shall be concluded that $L_i(t+1) \geq L_i(t) > \text{Load}_1(t)$.

Hence for all the cases it has been shown that $L_i(t+1) \geq \text{Load}_1(t)\ \forall\ i \in V$ and $\forall t \in \mathbb{N}$. So minimum load at t+1 is greater than minimum load at t which is given by $\text{Load}_1(t+1) \geq \text{Load}_1(t)\ \forall t \in \mathbb{N}$

Since $\left(\text{Load}_1(t)\right)_{t \geq 0}$ is an increasing order sequence and upper bounded then there exists a non negative

integer t_0 such that $\text{Load}_1(t) = \text{Load}_1(t^1) \forall\, t \geq t^1$. So part (I) of lemma is proved for any time $t \geq t^1$.

To prove part (II) of Lemma1, Let $V_1(t) = \{i \in V | L_i(t) = \text{Load}_1(t)\}$ is the set of nodes with load value less than or equal to the average load of the system at time t and it will be seen that $V_1(t^1) \supseteq V_1(t^1 + 1) \supseteq V_1(t^1 + 2) \supseteq \cdots$, i.e., the sequence of sets of nodes with minimum load is a decreasing sequence beyond t^1.

Let $t \geq t^1$ and $i \in V \backslash V_1(t)$. It will be seen that $i \notin V_1(t+1)$, i.e., if node i is not a node with minimum load value at time t then it will be never be a node having a minimum load value at time t+1.

- If $t \notin T_i$, then it has been proven that $L_i(t+1) \geq L_i(t) > \text{Load}_1(t) = \text{Load}_1(t+1)$. *so* $i \notin V_1(t+1)$.
- If $t \in T_i$, some different cases can be found:

Case 1: If $\varphi_i(t) > 0$ then, as it has been observed $L_i(t+1) > Load_1(t) = Load_1(t+1)$. So $i \notin V_1(t+1)$.

Case 2: If $\varphi_i(t) = 0$ and node i is an underloaded node ($\text{deviation}_i(t) < 0$), then it has been seen that $L_i(t+1) \geq L_i(t) > Load_1(t) = Load_1(t+1)$ thus $i \notin V_1(t+1)$.

Case 3: If $\varphi_i(t) = 0$ and node i is an overloaded node ($\text{deviation}_i(t) \geq 0$), then it has been seen that $L_i(t+1) \geq \bar{L}_i(t) > \text{Load}_1(t) = \text{Load}_1(t+1)$ and $i \notin V_1(t+1)$.

So it can be concluded that $V_1(t^1) \supseteq V_1(t^1 + 1) \supseteq V_1(t^1 + 2) \supseteq \cdots$.

Since $V_1(t^1)$ is a finite set, there exists an integer $t_1 \geq t^1$ such that $V_1(t) = V_1(t_1) \forall\, t \geq t_1$.

Note that if *node* i has the minimum load value at time $t_1 (\, i \in V_1(t_1))$ then $\varphi_i(t) = 0 \;\forall\, t \geq t_1$. Because it is an under loaded node($deviation_i(t) \leq 0 \;\forall\, t \geq t_1$) , it does not send any load such that $x_{ij}(t) = 0, \forall\, j \in V, \forall t \geq t_1$ and does not receive any load from neighbors.

Since both (I) & (II) of Lemma 1 is proved, hence Lemma 1 is proved

5. SIMULATION

The objective for load balancing is obtain a uniform response time throughout the system. For evaluating the performance of the load balancing algorithms M/D/1 Queuing model is used. The expected response time of a node is modeled as a M/D/1 queue depends on the mean and standard deviation of service times and on the average inter-arrival time. The D in this model stands for constant service time. The inter arrival time of requests on a node has a mean value of which follows a Poisson distribution

5.1 INFLUENCE OF NODE SIZE ON EXECUTION TIME OF AN ALGORITHM

To test the influence of node size on the execution time of an algorithm, the loads are varied from light load situations to high load situations by adjusting the parameter λ. The sizes of the nodes are varied from 100 nodes to 500 nodes to test the scalability of the algorithms. It has been observed under light load conditions (λ=0.1 jobs/sec), the GDE algorithm has been performing well when compared to the other algorithms inclusive of the proposed algorithm. But under heavy load conditions(λ=0.9 jobs/sec), the proposed algorithm has done well compared to the existing algorithms. The graphs presented in below figures show the time taken to balance the loads among the nodes. It has been observed that the algorithm execution time increases considerably with the increase in the number of nodes. SID is scalable, but its variance is increasing considerably with the increase in the number of nodes.

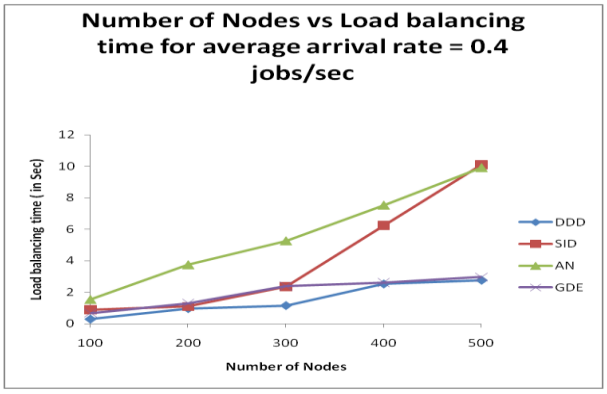


Figure 1: The effect of system size on load balancing time for different algorithms for average arrival rate=0.4 jobs/sec.

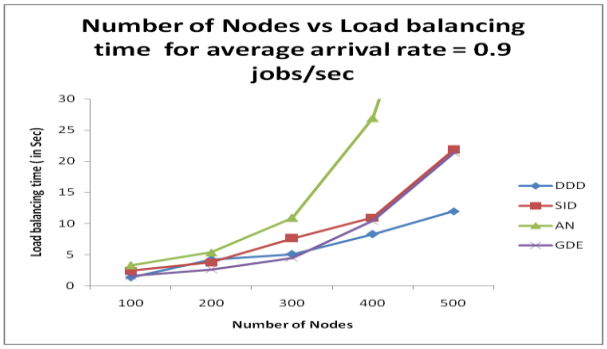


Figure 2: The effect of system size on load balancing time for different algorithms for average arrival rate=0.9 jobs/sec.

5.2 EFFECT OF NODE SIZE ON VARIANCE

To observe the maximum load difference between nodes in a distribute system as well as the variance among the nodes, the load balancing algorithm is run for the various job inter arrival patterns .At the end of load balancing algorithm, the variance among the nodes is computed. For moderate and heavy loads, the nodes have taken more time to reach the threshold level (L^{max}- L^{min} <=1). If the variance shows an irregular pattern it is an indication of job thrashing, where the same loads are shuttling between the nodes. But in proposed algorithm it shall be observed that irregular pattern is not

present and the variance among the nodes is decreasing constantly as the load balancing algorithm progresses over time.

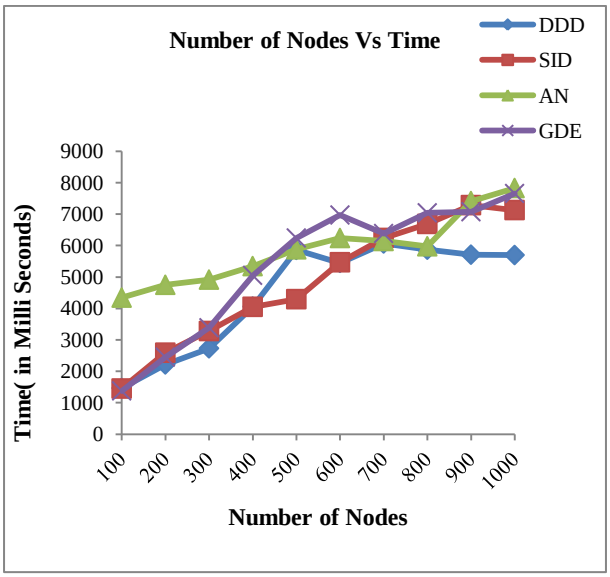
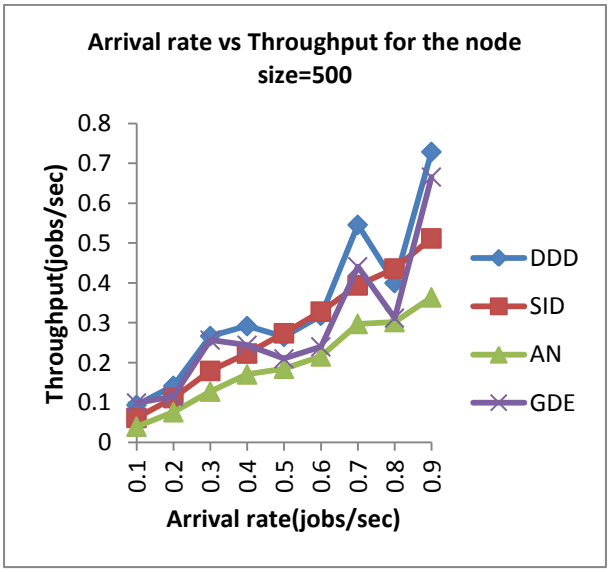


Figure 3 shows the time taken for different algorithms to reduce variance among the nodes in the system when the average arrival rate λ=10 jobs/sec.

5.3 EFFECT OF ARRIVAL RATE ON THROUGHPUT

In this simulation, the nodes are generated randomly with reachability of every node in the system. The edges connecting the nodes are generated in such a way that the graph is a spanning tree. In low load conditions, the GDE has performed well in executing the more number of jobs per unit time. Even though AN & the proposed algorithm DDD has attained low variance they have taken more time to reduce the variance among the nodes in the system thus resulting in low throughput when compared to GDE. But in moderate and heavy load conditions, the proposed algorithm has done well when compared to the other algorithms. The GDE algorithm has occupied next to the DDD algorithm.



6. REFERENCES

[1] J.E. Boillat, Load Balancing and Poisson Equation in a Graph, Concurrency: Practice and Experience, Vol. 2(4), December 1990, pp. 289-313.

[2] G.Cybenko, Load balancing for distributed memory multiprocessors. Journal of Parallel and Distributed Computing, 7:279-301, 1989.

[3] Rupali Bhardwaj, V.S.Dixit, Anil Kr.Upadhyay. A Propound Method for Agent Based Dynamic Load Balancing Algorithm For Heterogeneous P2P Systems in International Conference on Intelligent Agent and Multi-Agent Systems, 2009.

[4] F.M. auf der Heide, B. Oesterdiekhoff, and R. Wanka. Strongly adaptive token distribution. Algorithmica 15 (1996), pp. 413–427.

[5] Berenbrink, P. and Friedetzky, T. and Martin, R. (2005) 'Dynamic diffusion load balancing.', in Automata, languages and programming : 32nd International Colloquium, ICALP 2005, 11-15 July 2005, Lisbon, Portugal ; proceedings. Berlin: Springer, pp. 1386-1398.

[6] Cortés, A., Ripoll, A., Cedó, F., Senar, M. A., and Luque, E. 2002. An asynchronous and iterative load balancing algorithm for discrete load model. J. Parallel Distrib. Comput. 62, 12 (Dec. 2002), 1729-1746.

[7] Y.F. Hu, R.J. Blake, An Improved diffusion algorithm for dynamic load balancing, Parallel Computing 25(1999), pp. 417-444.

[8] E. Luque, A.Ripoll, A.Cortes and T. Margalef, A Distributed Diffusion method for dynamic load balancing on parallel computers,1995.

[9] Tina A. Murphy and John G. Vaughan, On the Relative Performance of Diffusion and Dimension Exchange Load Balancing in Hypercubes, Procc .of the Fifth Euromicro Workshop on Parallel and Distributed Processing, PDP'97, January 1997, pp. 29-34.

[10] P. Berenbrink, T. Friedetzky, and Z. Hu. A new analytical method for parallel, diffusion-type load balancing. J. Parallel Distrib. Comput., 69(1):54–61, 2009

[11] F. Cedo, A. Cortes, A. Ripoll, M. A. Senar, and E. Luque. The convergence of realistic distributed loadbalancing algorithms. Theor. Comp. Sys., 41(4):609– 618, 2007.

[12] D.P. Bertsekas and J. Tsitsiklis, Parallel and Distributed Computation: Numerical Methods,Prentice-Hall, Englewood Cliffs, NJ, 1989.

[13] Liu, J., Jin, X. and Wang, Y. 2005. Agent-Based Load Balancing on Homogeneous Minigrids: Macroscopic Modeling and Characterization, IEEE Transactions on Parallel and Distributed Systems, 586-594.

[14] Raghu Subramain, Issac D. Scherson, An Analysis of Diffusive Load-Balancing. In Proceedings of 6th ACM Symposiummon Parallel Algorithms and Architectures, 1994.

[15] T. Friedrich and T. Sauerwald, "Near-perfect load balancing by randomized rounding", in Proc. STOC, 2009, pp.121-130.

[16] Marc H. Willebeek-LeMair, Anthony P. Reeves, Strategies for Dynamic Load Balancing on Highly Parallel Computers, IEEE Transaction on Parallel and Distributed Systems, vol 4, No 9, September 1993, pp.979-993.

[17] Qiao, Y. and Bochmann, G. v. 2009. A Diffusive Load Balancing Scheme for Clustered Peer-toPeer Systems. In Proceedings of 15th ICPADS. IEEE Computer Society, 842-847

[18] S. Muthukrishnan, B. Ghosh, and M. Schultz. First and second-order diffusive methods for rapid, coarse,distributed load balancing. Theory of Computing Systems, 31(4):331–354, 1998

[19] Luling, R., Monien, B. 1993. A Dynamic Distributed Load Balancing Algorithm with Provable Good Performance. Proc. of the 5th ACM Symposium on Parallel Algorithms and Architectures, 164-173

[20] Acker, D., Kulkarni, S. 2007. A Dynamic Load Dispersion Algorithm for Load Balancing in a Heterogeneous Grid System. IEEE Sarnoff Symposium, 1- 5.