International Journal of Distributed and Parallel Systems (IJDPS) Vol.3, No.1, January 2012

DECENTRALIZED LOAD BALANCING IN HETEROGENEOUS SYSTEMS USING DIFFUSION APPROACH

P.Neelakantan

Department of Computer Science & Engineering, SVCET, Chittoor

pneelakantan@rediffmail.com

ABSTRACT

The grid and cluster computing uses interconnected nodes to solve a problem in parallel in order to improve the response time of the system. Diffusive load balancing algorithms works well when the nodes in the system have the same processing capacity. But little attention was paid in diffusion load balancing algorithms in the literature for distributing the workload in the nodes with different processing capabilities when the load of the nodes is treated as integers. When the loads are distributed to the nodes without considering their processing capacities it would affect the response time of the system. Effective load balancing in heterogeneity can be achieved by considering the processing capacities of the nodes. This paper propose a diffusive load balancing algorithm which distributes a proportion of excessive workload of heavily loaded node to lightly loaded node by considering the nodes processing capabilities.

1. INTRODUCTION

Grid computing and cluster computing are examples of distributed computing which consists of many heterogeneous nodes with different processing capacities connected by a communication network. Load balancing must be used when the load among the nodes are not uniform. The nodes in a distributed system must have the uniform distribution of loads after applying the load balancing algorithm. For migrating the load from the heavily loaded node to lightly loaded node in a heterogeneous distributed system, their processing capacities must be considered. So, the load balancing algorithms designed for homogeneous distributed systems won't work efficiently for the heterogeneous systems. So, there is a need for load balancing algorithm which distributes the load among the nodes in a heterogeneous system by taking into consideration of the processing capacity of the nodes. Distributed systems are geographically spread over different autonomous users and organizations which make them potentially large [4].

Load balancing algorithms are divided into static and dynamic load balancing algorithms. Static load balancing algorithms do not base their decision on the current state of the system, so, their performance would not be optimal when compared to the dynamic load balancing algorithms which base their decision on the current state of the system. But dynamic load balancing algorithms incurs more overhead when compared to the static load balancing algorithms. Dynamic policies are further divided into two classes: Centralized and Distributed [1, 2,3]. In centralized polices one node is held responsible for maintain the information of all the nodes in the system.

Collecting the information from all the nodes and then distributing the work load among the nodes would be the difficult task and it imposes much overhead and it won't perform well when the scalability of the system increases. In distributed policies, each node collects the information from all nodes to know whether it is overloaded or underloaded. A dynamic distributed load balancing algorithm has three components: (1) location policy determines suitable nodes for exchange of the load(2) Information policy is used to collect the load information from the nodes in the system(3)transfer policy determines , whether the node is suitable candidate for transfer of load.

Even this method is not feasible when the number of nodes in the distributed computing is large. The distributed load balancing schemes are further classified into sender –initiated and receiver- initiated. Iterative load balancing schemes are found to be efficient in balancing the work load among the nodes. Here the nodes will collect information from only the neighhors which reduce the communication overhead and also to achieve faster convergence. In diffusion approach, the node can exchange the information from all its neighbors in single communication step. In literature there is no diffusion load balancing algorithm that deals with heterogeneous systems by considering the load as an integer quantity. In this paper a decentralized load balancing algorithm has been devised for heterogeneous systems by using diffusion approach.

2. MATHEMATICAL MODEL

The network is represented by an undirected graph G=(V ,E), where each node $i \in V$ has a processing weight $p_i > 0$, and a processing capacity $c_i > 0$ and E $\subseteq V \, X \, V$ is a set of edges represents the communication links connecting the nodes in the heterogeneous distributed system.

N: Number of nodes in a system
V= {1, 2,….n} represents set of nodes in a system
DL: Deficit Load
N_i: Neighbors of the node i denoted by N_i={$j | \, j \in V \, and \, (i,j) \in E$}
P_i: processing weights of node i
C_i: processing capacity of node i
j: Index of the neighbouring nodes
DN_i: Set consisting of deficient neighbours belonging to domain of node i
L_i: Load of node i
$Nless$: Set consisting of the nodes having the same minimum load value
δ_1, δ_2: Indicators of apportion of load sent to the deficient neighbors.

The load of node i can be defined as $L_i = \frac{P_i}{C_i}$. When two nodes are connected by an edge, then they are neighbors to each other and the two nodes can exchange their loads (task weight) through that edge. To minimize their loads the node transfers their excess load to their neighbors. In general, it is not feasible for a node to collect the load information from all the nodes in the network. Therefore, in order to reduce the communication delays, in our model it is assumed that a node knows the load information of its neighbors only. The loads of other nodes are unknown. A network is referred to as homogeneous network when the processing capacities of all nodes are equal; otherwise it is referred to as heterogeneous network. In homogeneous systems, the value of C_i= 1, and therefore $l_i = P_i$.

3. PITFALLS IN DESIGNING LOAD BALANCING ALGORITHM

The algorithm that work in decentralized manner to balance the loads among the nodes in a heterogeneous distributed system has to focus on three aspects which have impact on the convergence rate and number of steps required to reach convergence.

1. Choosing target nodes
2. Amount of the excess load to be sent
3. Distribution of the excess load

The first aspect aims on nodes to be considered in domain of node i that invoked the load balancing algorithm to send its excess load. Algorithms that belong to decentralized category will know the load information of only its neighbors in its domain, so, it is obvious that it will send load to its neighbors. But when neighbour node consists of more load than the node that is trying to send its excess load, then that neighbour node is not considered for sending the load.

The load is sent only to the deficit neighbours and deficit neighbours would be identified in a domain of node *i* that has invoked the load balancing algorithm. To do this the deficit neighbours are chosen in such a way that the load value of the deficit neighbours must be less than the load value of the node that is trying to send apportion of its excess load.

The second aspect concerns about how much load is to be removed and sent to the neighbors. The amount of the load to be sent depends on how much the current node is overloaded with respect to its neighbors.

Once it has been known the amount of load to be removed from the overloaded node to be sent to the target deficit neighbours, then there are different ways to deal with third aspect. The first way is to divide the load among the deficit neighbors that receives the load. The second way is node i to distribute its entire excess load to one deficit neighbour of all chosen deficit neighbors in its domain.

3.1 Choosing the target nodes

Since the goal load balancing algorithm is to balance the load, obviously heavier nodes should send load to lighter nodes. In order to do this, node i has to select a subset of neighbors N_i, where$(N_i = \{j \in N_i | \ L_j < L_i\})$ .The algorithm runs in a decentralized manner in all the nodes, so nodes would send their excess load to lighter nodes to achieve balance among the nodes.

The use of neighbourhood information can be further improved to estimate the average load of the local domain. In order do this, the load information of all neighbor nodes (i.e., nodes connected through an edge to node $i)$ is collected. Then node i calculates the load average, which is given by $\overline{L_\iota}=\left(\frac{l_i+\sum_{j\in N_i} l_j}{(|N_i+1|)}\right)$. If the load of node i is greater than $\overline{L}$, then it is overloaded, so it has send load to the nodes which have load $L_j < \overline{L}$. This strategy can be used to achieve faster global load balancing.

3.2 Amount of the excess load to be sent

The amount of the load to be moved among deficit neighbors would have direct impact on the convergence of the algorithm. If an algorithm is designed in such a way that allows sending small amounts of load, the nodes in the network takes long time to converge depending on network topology, but the stability would be greater. On the other side, if algorithm allows the nodes to send large proportions of load, then the convergence would be faster but the system would not be stable and even balancing could not be reached which has a contrary effect on the convergence speed.

Once the subsets of neighbors have been defined, the algorithm has to calculate how much load is to be sent to the deficit neighbors. In general the amount of the load to be moved from the overloaded node to the deficit neighhors would be the difference of load of overloaded node i and the average load of domain of node i which is given by $\delta = L_i - \overline{L}$.

If δ load is sent to each node in the subset of neighbors, then each node receives a load of $\frac{\delta}{|DN_i|}$ to each node in N_i, where $|N_i|$ indicates the number of nodes in N_i. This approach would have an advantage of being fair and simple as all nodes receives the same load and no node is privileged. Another approach is to send δ load only to the least loaded node(s). In this case the convergence rate becomes faster as the least loaded nodes are filled quickly. But they would become target to his neighbors and results in accumulation of load rather than required. Both of these approaches have their own advantage and disadvantages and none of them prove adequate.

3.3 Sending load to the deficit neighbours

When an algorithm is designed in such a way that allows overloaded node to send small amount of load to the other nodes in the network it takes long time to converge. The time that it takes to converge depends on the network topology, but those algorithms will have better stability with the number of nodes increasing in a system. On other hand, if an algorithm is designed to send large portions of the load to the deficit neighbours in a domain, the convergence would be faster but the system would not be stable and no node in the system reaches equilibrium (all the nodes reach equilibrium when all the nodes have equal load) which has a adverse effect on the convergence speed.

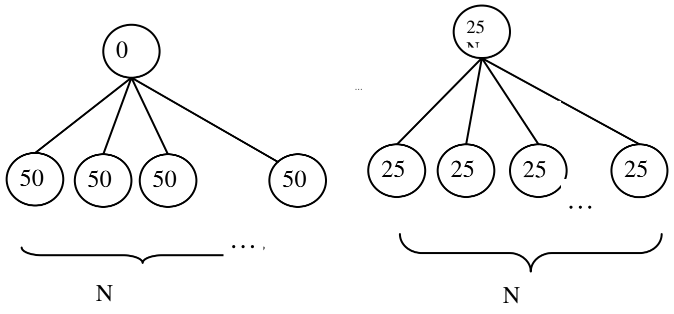


Figure 1: when loads are moved in small proportion to the deficit neighbors

To analyze the above techniques, let consider in Figure 1, the nodes are sending loads cautiously where the load fluctuations could not occur and all nodes reach stable state slowly. A node is said to be reached to the stable state when the loads of nodes are equal to the average load of the system. Another approach is to distribute half of the loads to the deficient neighbors which avoid load fluctuations and also faster convergence is achieved but it would not work for few networks as shown in below figure 2.

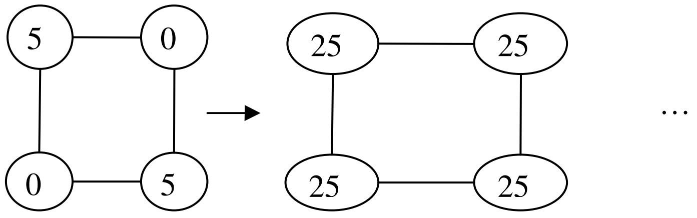


Figure 2: A network showing one deficit neighbour targeted by all loaded nodes.

As seen in the above figure every node attempts to send half its load to only one deficit neighbour and the deficit neighbour with load value equal to zero becomes 50* N and it is the node in the network which is heavily loaded and it takes O(N) time to make the system into stable state. But when below topology is considered as shown in the figure 3 and the same logic is applied, the nodes in the system reaches stable state in a single step.

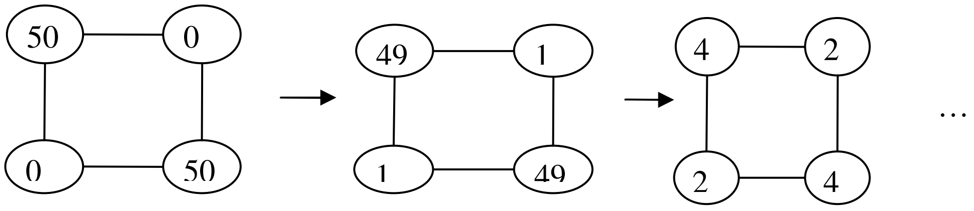


Figure 3: four nodes with two nodes heavily loaded and two nodes with zero load.

4 Proposed method

Keeping in view of the above mentioned problems, the algorithm *DISHET* has been proposed which uses a diffusion approach for sending the loads to the deficient neighbors. This algorithm is run in each node i.

4.1 Description of the proposed method

In a proposed method, a set of deficit neighbors of domain of node i is given by

$$DN = \{j \in N_i \mid L_j < L_i\} \rightarrow (1)$$

After identifying the deficit neighbors of domain of node i, the load average domain of node i is calculated by using the formula

$$L_{avg} = \frac{P_i + \sum_{j \in DN} P_j}{C_i + \sum_{j \in DN} C_j} \rightarrow (2)$$

Note in the above equation the processing capacities of the nodes are taken into account which is true in heterogeneous computing. In general if node i take t_i time units to process a task t and node j takes t_j time units to process the same task t then the ratio of t_i/t_j must be equal to one. Otherwise the ratios are different. The former is the case, when the nodes have different processing capacities and the latter refers to when nodes have same processing capacities.

After computing the load average of the system, the load difference in the domain of node i is calculated by using the formula

$$\text{LD}= (L_i\text{-}L_{avg})\, C_i \rightarrow (3)$$

When the value of LD is positive, then it indicates the node i is having an excess load, and it is to be sent to the deficit neighbors belong to the set DN_i. C_i indicates the relative processing capacity of node i when compared to other nodes in the system. If the load difference is negative value the node i do not execute the *procedure Balance* otherwise it calls the *procedure Balance.*

In procedure Balance the input parameters are the load difference and the set of deficit neighbors. The algorithm is terminated when the load difference LD is reached to 1. First the excess load is to the least loaded node in the set deficit neighbour DN_i. For this purpose the nodes in set DN_i are sorted in ascending order of their load values which takes time complexity of O(d log d), where d is the number of nodes in the deficit neighbors(d=| DN_i |). After sorting the nodes according to their load values in set DN_i, the first node is considered for migrating the load and also a check is made to see if any node contains load value of the first node. If it is those elements are kept in the set *nless* by incrementing the counter value and for each element in the set DN_i, the following is done:

```
    $for\ \ k \in \{DN_i\}$
  $if\ (L_{Lowload} == L_k)$
        $nless = nless \cup \{k\};$
        $count = count + 1;$
  $endif$
```

The T*empset* consists of the nodes which is present in DN_i but not in *nless*

$$\text{Tempset} = DN_i - nless \rightarrow (4)$$

The load to be sent is based on Tempset and how much to be sent is determined by use of two indicators δ_1 and δ_2 .The minimum of these two values have been used for sending the load to the deficit neighbors until the value of load difference LD reaches zero.

$$\delta_1 = \frac{LD}{count} \ \ \&\delta_2 = L_k - L_L \ \ \text{Where k} \in \text{tempset and L} \in \text{nless} \rightarrow (5)$$

When the load index of the neighbouring nodes falls below the average load of the domain of $node$ i, it is said to be deficient neighbour. The deficient neighbours do not have the excess load to send and they receive apportion of the load sent by the excessive neighbors. If all the loads of the nodes are equal to the average load of the underlying domain, then the domain is in balanced state.

```
Algorithm DISHET ()
        Begin for node i
            $DN_i=\{j \in N_i | \, L_j < L_i\}$
            Let $L_{avg} = \frac{P_i + \sum_{j \in DN} P_j}{S_i + \sum_{j \in DN} S_j}$;
            LD= $(L_i\text{-}L_{avg})\, S_i$;
            if (LD<0) then exit;
            Balance (LD, $DN_i$);
        End

  Procedure Balance (LD, $DN_i$ )
        Begin
         While (LD>1)
           // sort the loads in ascending order in set $DN_i$
           $Lowload = FirstElement\ in\ DN_i$;
           $count = 0$
```

```
            $Nless = \emptyset$
            $for\ \ k \in \{DN_i\}$
            $if\ (L_{Lowload} = L_k)$
                    $nless = nless \cup \{k\};$
                    $count = count + 1;$
                $endif$
          $end\ for$
            $Tempset = DN_i - nless;$
          $\delta_1 = \frac{LD}{count};$
            $if(\ tempset \neq \emptyset)$
                    $Nestloadmin = firstelement\ in\ Tempset$
                    $\delta_2 = L_{nextloadmin} - L_{lowload};$
                $for\ each\ \ k\ \in\ nless$
                    $if(\delta_1 > \delta_2)$
                        $Transfer\ \delta_2\ to\ k;$
                        $LD = LD - \delta_2;$
                    $end\ if$
                    $else$
                        $Transfer\ \delta_1\ to$
                        $LD = LD - \delta_1;$
                    $end\ else$
                $end\ for$
            $end\ if$
        $else$
        $for\ each\ k \in nless$
                $Transfer\ \delta_1\ to\ k;$
            $LD = LD - \delta_1;$
        $End\ for$
        $End\ else$
    End while
End
```

Algorithm: Balance

The above algorithm uses diffusion technique for balancing the load among the nodes in the distributed system. This algorithm receives load information from the neighbours as an input. The idea behind this algorithm is to distribute load equally among the lightest nodes in DN_i until their load value reaches the second lightest node in DN_i. This process is repeated until the value of DL runs out.

Lemma 1: Let L_i^t be the load of node i at time t. Let L_i^t = ($L_1^t, L_2^t, \ldots\ldots L_n^t$) be the array of the loads in time t sorted in ascending order. If there exists at least one node with DL> 0 in time t then L_{t+1} is lexicographically greater than L_t .

Proof: At time t some nodes may send their load to the neighboring nodes may have their load decreased in time $t + 1$; therefore these nodes do not send the load at time $t + 1$. This refers to the case when loads of the nodes are distinct. But by using the same idea a similar proof shall be derived where all loads are not distinct.

Let S be the set of nodes that send load (i.e., the nodes with DL > 0) in time t, Let$S_i^t \neq \emptyset$. Let the node does belong to S_i^t shall also receive load in time t. Let γ =$Min_i\{l_i^{t+1}: i\ \in S\}$. That is, γ is the node having lowest load value at time $t + 1$ among the nodes that sent load in time t.

Let γ occupies the k-th position of the array L_{t+1}.Let Q_t = ($L_1^t, L_2^t, \ldots\ldots L_{k-1}^t$) be the array of loads in first k-1 positions of Q_t . It has been seen that a node i belongs to Q_t if its load is among the load values of the array Q_t . Nodes belonging to Q_{t+1} will receive load (without sending) or remained unchanged in time t will depend on γ. Thus, all the load values in Q_{t+1} are greater than or equal to the corresponding values of Q_t .

Next, let us consider two cases.

i. A set Q_{t+1} contains a node i that received load in time t. In this case, node i belongs to both Q_t and Q_{t+1}, and its load value is increased in time t + 1. Therefore, there will be at least one load value in set Q_{t+1} strictly greater than one value in Q_t and therefore L_{t+1} is lexicographically greater than L_t .
In the complementary case, it has been seen that

ii. There are nodes belonging to Q_{t+1} which do not send or receive the load in time t. Thus, the load values in set Q_t and Q_{t+1} are equal. In this case, it has been shown that the load value at kth position at time t+1 is strictly greater than the load value in the same position at time t which has received load from S. Let r be node with less load value in time t which received load from S. As r received load, it cannot belong to Q_t . Thus, in time t, r would be at least in the k-th position. Therefore, the value of the $k-th$ position of L_t is at most l_r^t. Note that $L_\gamma^{t+1} \geq L_\gamma^t - DL = \overline{l_\gamma^t} > l_s^t$ where $\overline{l_\gamma^t}$ is the mean value for node q in time t and the last inequality is true because r received load from γ in time t. So, the value of the k-th position increased, and therefore $L_{t+1}$is lexicographically greater than L_t. Note that this proof is valid for any number of nodes which sends the excess load in a given time t, therefore it is valid for the asynchronous case.

Theorem 1 Convergence: In asynchronous and heterogeneous networks, if the nodes use the algorithm DISHET, then the system converges to a balanced state.

Proof: If the nodes in the heterogeneous distributed systems are not balanced exhibits that there is at least one node in the system heavily loaded, let i be the most loaded node. By the choice of i, all $j \epsilon N_i$ have $L_j \leq L_i$. Moreover, as i is not balanced, then at least one $j \epsilon N_i$ has $L_j \leq L_i$. When calculating $\overline{L_\iota}$ the nodes $j \epsilon N_i$ such that $(\overline{L_\iota})=\frac{P_i+\sum_{j \in DN_i} P_j}{c_i+\sum_{j \in DN_i} c_j}$ have their load value rounded to L_i. But if $L_i > \overline{L_\iota}$ and consequently, $\delta = L_i - \overline{L_\iota} > 0$. Thus, the result of Lemma 1 shall be used, which guarantees that the vector of loads sorted in ascending order, in the next time moment, is lexicographically greater than the array of the current step.

Let us assume algorithm DISHET is executed by some of the nodes in the system asynchronously in a given time instant t. Let $S \subseteq V$ be the set of nodes executed the algorithm DISHET and nodes have changed their load value by executing the algorithm in time t.

Let L be the array of loads sorted in ascending order of load values in time t. It has been shown that L_{t+1} is lexicographically greater than L_t . Let $S_{min} \subseteq S$ be the lightly loaded nodes in a time t. There exists at least one node $v \in S_{min}$ which is adjacent to node k such that $l_k^t > l_v^t$. Now by using the algorithm DISHET, node k sends a portion of its load to the neighbor node v but the node v does not send any load in time t because it is underloaded when compared to the average load of the domain.

In time $t+1$ the load value of node k in $S \setminus S_{min}$ decreases but its load value never becomes smaller than the load value of node which is given by $L_k^{t+1} \geq L > L_v^t$. Thus L_{t+1} is lexicographically greater than L_t. Thus it has been proven that the sorted array of load values of nodes in time t in S are lexicographically greater than the sorted array of load values of nodes in S at time $t+1$.

Lemma 3: A node i can only send load to a neighbor j if $L_i > L_j, (L_j - L_i > \varepsilon)$ i.e., where $j \in N_i$, N_i is the neighbouring nodes (directly connected to the node i)

In other words, this rule ensures that if loads of two nodes belong to same domain differs significantly, then load will be transferred between two nodes belonging to same domain.

Lemma 4: Let L_i^t be the load of node i at time t. Then

- $L_j^t \approx \overline{L_\iota^t}$
- $L_i^t \geq L_j^t$

The first constraint says that the load on a nodes in a domain of node i does not deviate too much when compared to the load average of the domain of node i and the second one guarantees, the load balancing algorithm must distribute the load to the deficit neighbour in such a way that, the node after transferring its excess load to its neighbour, its load value must remain greater or equal to its deficit neighbour. The second one is a technical constraint because to avoid task thrashing effect.

5. Simulation

The proposed load-balancing algorithm DISHET is compared to the two heterogeneous load balancing algorithms: Random biasing and Dynamic Biasing .The simulation framework has been designed including random graph which range from 8 to 64 nodes and for different load distributions patterns which vary from situations which exhibit a light unbalance degree to high unbalance situations. The comparison has been carried out in terms of number of iterations, throughput and average response time for the varying loads.
For that purpose, Load Balancing Simulator in Java which allows us to:

- Test the behaviour of different load balancing algorithms under the same conditions
- Evaluate the behaviour of the load balancing algorithms for random graph with different sizes of nodes
- Evaluate the behaviour of the algorithms for different load situations
  The load distribution among the nodes are carried in the following manner

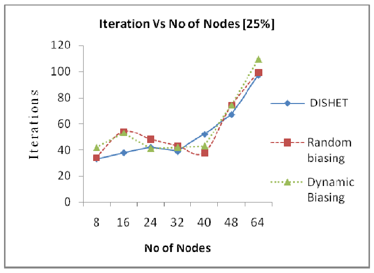
  - Initial load distributions varying 25% from the global load average
    - $\forall i\ L_i(0) \in$ [L/n – 0.25*L/n, L/n+0.25*L/n]
  - Initial load distributions varying 50% from the global load average
    - $\forall i\ L_i(0) \in$ [L/n – 0.50*L/n, L/n+0.50*L/n]
  - Initial load distributions varying 75% from the global load average
    - $\forall i\ L_i(0) \in$ [L/n – 0.75*L/n, L/n+0.75*L/n]
  - Initial load distributions varying 100% from the global load average
    - $\forall i\ L_i(0) \in$ [L/n –L/n, L/n+L/n]

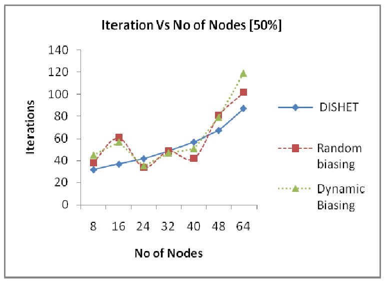
The 25% variation pattern corresponds to the situation where all nodes have similar load at the beginning and these loads are close to the global average. i.e., the initial situation is quite balanced. On the other hand, the 100% variation pattern corresponds to the situation where the difference of load between nodes at the beginning is considerable. 50% and 75% variation patterns constitute intermediate situations between the other two.
The group of pathological distributions was also used in order to evaluate the behaviour of the strategies under extreme initial distributions. In these distributions a significant amount of nodes has a zero initial load. These scenarios seem less likely to appear in practice, but we have used them for the sake of completeness in the evaluation of the strategies. The pathological distributions were classified in four groups:

- Initial load distribution, where all the load is located on a single node :L(0) = {L,0,…0} , i.e., there are n-1 idle nodes
- 25% of idle nodes, a quarter of the nodes have an initial load equal to 0.
- 50% of idle nodes, a quarter of the nodes have an initial load equal to 0.
- 75% of idle nodes, a quarter of the nodes have an initial load equal to 0.

In our simulations, the problem size is known beforehand and all the experiments included in this in this paper are performed for a varying Load sizes equal to 1000, 2000, 4000 ,6000,8000,10000 load units. Therefore, the expected final load at each processor, i.e., the global load average, can be evaluated a priori to be $\lceil L/n \rceil$ or $\lfloor L/n \rfloor$, n being the size of the nodes in the heterogeneous systems.

Graphs for normal load distribution where load values are varied from 25% to 100 %.





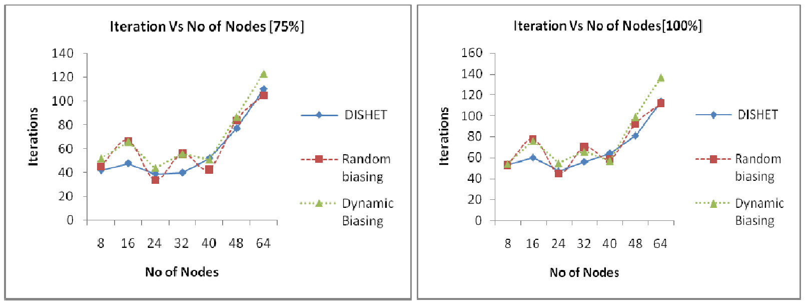


Figure-1: Iteration Vs No. Of Nodes

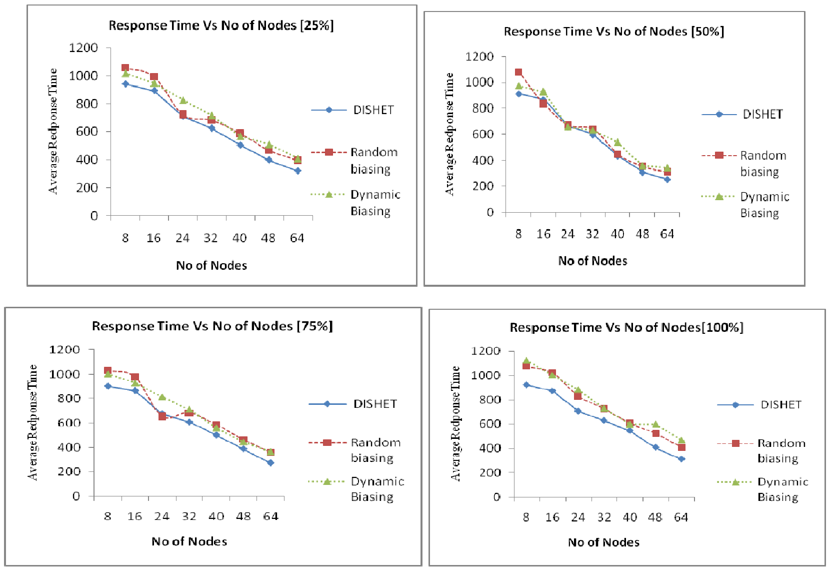


Figure-2: Response Time Vs No. of Nodes

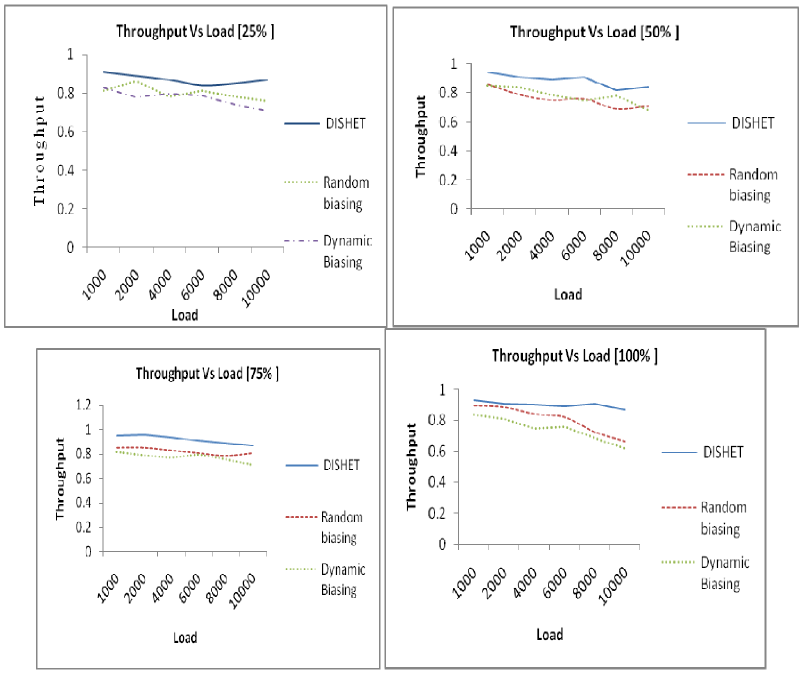


Figure-3: Throughput Vs Load

Graphs for spike load distribution

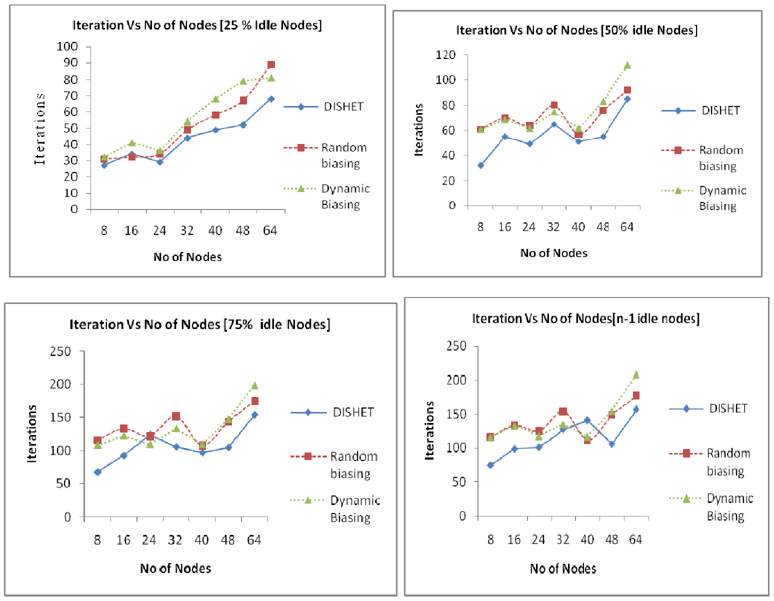


Figure-4: Iteration Vs No of Nodes

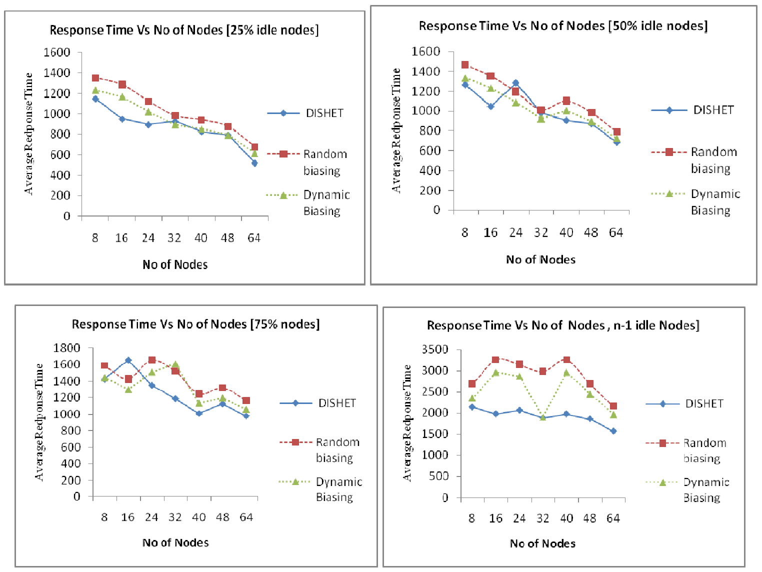
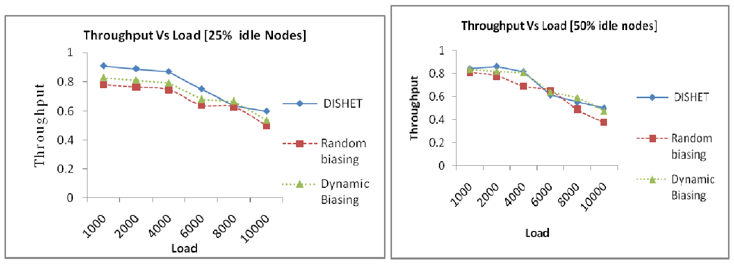


Figure-5: Response Time Vs No of Nodes



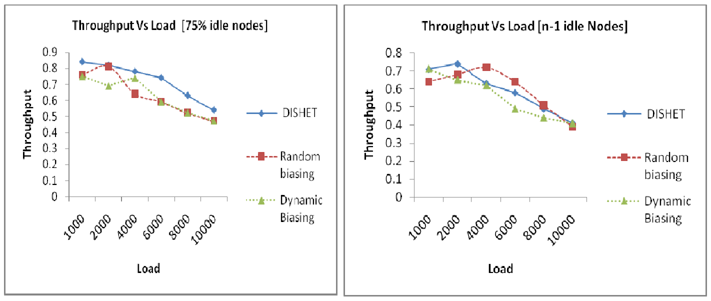


Figure-6: Throughput Vs Load

REFERENCES

[1] T.Casavant and J.G.Kuhl.A taxonomy of scheduling in general-purpose distributed computing systems, IEEE Trans. Software Eng.,14(2):141-154,February,1988.

[2] X.Tang and S.T.Chanson,Optimizing stiatic job scheduling in a network of heterogeneous computers,In Proc of the International conference on parallel processing, pages 373-382,August,2000.

[3] V.Bharadwaj and G.Barlas,Efficient scheduling strategies for processing Multiple divisible loads on Bus networks,Journal of parallel and distributed computing, vol.62,no.1,pp.132-151,Jan 2002.

[4] O.Akay and K.Erciyes, "A dynamic load balancing model for a distributed system",Mathematical and computational applications, vo.8(1-3),pp.353-360,2003.

[5] D. Z. Gu, L. Yang and L. R. Welch, A Predictive, Decentralized Load Balancing Approach, in: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, Denver, Colorado, 04-08 April 2005.

[6] V. Berten, J. Goossens, and E. Jeannot, On the distribution of sequential jobs in random brokering for heterogeneous computational Grids, IEEE Transactions on Parallel and Distributed Systems, 17 (2) (2006) 113-124.

[7] Lie,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,586-598

[8] Kai Lu, Riky Subrata, Albert Y. Zomaya. Towards Decentralized Load Balancing in a Computational Grid Environment. In Proceedings of GPC'2006. pp.466~477

[9] Luling,R., Monien,B.1993.Adynamic distributed load balancing algorithm with provable good performance. Proc of the 5th ACM symposium on parallel algorithms and architecture 164-173.

[10] MurataY.,Takizawa, H.,Inaba,T.and Kobayashi,H.2006,A distributed and cooperative load balancing mechanism for large scale p2p systesm.Proc International Symposium on applications and internet workshops 126-129

[11] Shah,R.,Veeravalli, B.and Misra M 2006,Estimation based load balancing algorithm for data –intensive Heterogeneous grid environments. Proc.13th International conference on High Performance Computing(HiPC'06),72-83.

[12] S. Zhou, A trace-driven simulation study of dynamic load balancing, IEEE Transactions on Software Engineering, 14 (9) (1988) 1327–1341.

[13] W. Zhu, P. Socko, B. Kiepuszewski, Migration impact on load balancing—an experience on Amoeba, ACM SIGOPS Operating Systems Review, 31(1) (1997),pp:43–53.

[14] D.L. Eager, E.D. Lazowska, J. Zahorjan, The limited performance benefits of migrating active processes for load sharing, ACM SIGMETRICS Performance Evaluation Review, 16 (1) (1988) 63–72.

[15] G. Cybenko, Dynamic load balancing for distributed memory multi-processors, Journal of Parallel and Distributed Computing, 7 (1989) 279–301.

[16] X. Qian, and Q. Yang, Load balancing on generalized hypercube and mesh multiprocessors with LAL, in: Proceedings of 11th International Conference on Distributed Computing Systems. 20-24 May 1991, pp. 402 –409.

[17] Barazandeh, I.; Mortazavi, S.S.; Two Hierarchical Dynamic Load Balancing Algorithms in Distributed Systems,2009, ICCEE '09. Second International Conference on computer and electrical engineering, pp 516-521.

[18] Hyo Cheol Ahn, Hee Yong, Kyu Yeong jeon, Dynamic load balancing for large-scale distributed systems with interlligent fuzzy controller,IEEE International conference on Information reuse and Integration. Aug 2007, PP 576-581