Distributed Systems – Emergence of Adaptive Load Sharing Method Using Hierarchical Policy

D.Sharanya1
M.Tech CSE Dept., Institute of Aeronautical Engineering, HYD-500043, TS, India
Dr.N. Chandra Sekhar Reddy2.
Professor, CSE Dept., Institute of Aeronautical Engineering, HYD-500043,TS, India
Baswanth3
Professor, CSE Dept., Institute of Aeronautical Engineering,HYD-500043, TS, India.

***Abstract*—we can improve the performance of distributed systems by load sharing (i.e., distribute load from heavily loaded nodes to lightly loaded ones). Adaptive load sharing policies take system state into account in making job distribution decisions. We can maintain the state information in one of two basic ways: distributed or centralized. This paper discuss on distributed systems which rely on either sender-initiated or receiver-initiated policies. While distribution of state information makes the distributed policies suitable for large distributed systems, they do suffer in performance. This paper mainly focuses on the adaptive load sharing on homogeneous distributed applications (Type-1). This paper also makes comparison over the Centralized and Distributed applications with different decision policies.**

***Index Terms*—Component, formatting, style, styling, insert.** *(key words)*

I. INTRODUCTION

Adaptive load sharing attempts to improve the performance of distributed application by sharing the resources in a dynamic or demand on resource policy. These load sharing policies can be either static or dynamic.

Static load sharing policies do not require system state information in making load distribution decisions where as the dynamic load sharing policies make their load distribution decisions based on the most recent or current system state. As the system state changes dynamically, dynamic load sharing policies tend to provide significant performance improvements compared to static policies and no load sharing.

This paper considers dynamic load sharing in heterogeneous distributed systems. As in [15], we consider two types of heterogeneous systems: type I and type II. In type I systems, all nodes are identical but the job arrival rate at different nodes can be different; in type II systems, processing rates of nodes are also different along with different job arrival rates.

Two important components of a dynamic policy are a transfer policy and a location policy [4]. The transfer policy determines whether a job is processed locally or remotely and the location policy determines the node to which a job, selected for possible remote execution, should be sent. Typically, transfer policies use some kind of load index threshold to determine whether the node is heavily loaded or not. Several load indices such as the CPU queue length, time averaged CPU length, CPU utilization, the amount of available memory etc. have been proposed/used [11,18,21]. It has been reported that the choice of load index has considerable effect on the performance and that the simple CPU queue length load index was found to be the most effective [11].

The dynamic policies can employ either a centralized or a distributed location policy. In the centralized policy, state information is collected by a single node (called the "coordinator") and all other nodes would have to consult this node for advice on the system state.

In the distributed policy, system state information is distributed to all nodes in the system. The centralized policy has the advantage of providing near perfect load sharing as the coordinator has the entire system state to make a load distribution decision. The obvious disadvantages are that it suffers from diminished fault-tolerance and the potential for performance bottleneck. In this context, it should be noted that some studies have shown that the node collecting and maintaining the system state need not be a performance bottleneck for reasonably large distributed systems [19]. However, if the system is geographically distributed (for example, several LAN clusters of nodes interconnected by a WAN), consulting a central node is very expensive and causes performance problems. Thus, the use of the centralized policy is often limited to a cluster of nodes in a large distributed system [22].

The distributed policy eliminates these disadvantages associated with the centralized policy; however, distributed policy may cause performance problems as the state information may have to be transmitted to all nodes in the system. Previous studies have shown that this overhead can be substantially reduced by sampling only a few randomly selected nodes [4,5]. A further problem with the distributed policies is that the performance of such policies is sensitive to variance in service times as well as inter-arrival times [1]. Distributed policies are, however, scalable to large system sizes.

To overcome these problems, we have proposed a hierarchical load sharing policy that combines the merits of the centralized and distributed policies while eliminating/minimizing the disadvantages of these policies [2]. The performance of the hierarchical policy in homogeneous distributed systems has been reported in [2]. In this paper, we focus on the performance of the hierarchical policy in *heterogeneous* distributed systems. Since distributed systems are often heterogeneous in nature and load sharing policies are sensitive to heterogeneity, it is important to study the performance in a heterogeneous system. The results reported here suggest that the hierarchical policy provides substantial performance improvements over the sender-initiated and receiver initiated policies; its performance is closer to that of the centralized policy while providing scalability and fault-tolerance closer to that of a distributed policy. Note that the centralized policy is the best policy if there is no contention/bottleneck problem for the coordinator.

Location policies can be divided into two basic classes: sender initiated or receiver-initiated. In sender-initiated policies, congested nodes attempt to transfer work to lightly loaded nodes.

In receiver-initiated policy, the lightly loaded nodes search for congested nodes from which work may be transferred. It has been shown that, when the first-come/first-served (FCFS) scheduling policy is used, sender-initiated policies perform better than receiver-initiated policies at low to moderate system loads [5].

This is because, at these system loads, the probability of finding a lightly loaded node is higher than that of finding a heavily loaded node. At high system loads, on the other hand, receiver-initiated policies are better because it is much easier to find a heavily loaded node at these system loads. There have also been proposals to incorporate the good features of both these policies [17,18].

The rest of the paper is organized as follows. Section2 reviews the existing load sharing techniques, Section 3 discuss the proposed method; section 4 analyzes the proposed load balancing technique over the existing techniques and finally concluded in section 5.

II. Related Work

This section deals with load sharing techniques which can be applied on homogenous and heterogeneous distributed system applications. They are broadly classified into two types. One-static and the second –Dynamic. The dynamic load sharing methods can be implemented by the following policies.

1. Sender-initiated Policy.
2. Receiver- Initiated Policy.
3. Single Coordinator Policy.
4. Hierarchical Policy.

A. Sender Initiated Policy

When a new job arrives at a node, the transfer policy described above would decide whether to place the job in the job queue or in the job transfer queue. If the job is placed in the job transfer queue, the job is eligible for transfer and the location policy is invoked. The location policy probes (up to) a maximum of probe limit *Pl* randomly selected nodes to locate a node with the job queue length less than *T*. If such a node is found, the job is transferred to that node for remote execution. The transferred job is directly placed in the destination node's job queue when it arrives. Note that probing stops as soon as a suitable target node is found. If all probes fail to locate a suitable node, the job is moved to the job queue to be processed locally. When a transferred job arrives at the destination node, the node must accept and process the transferred job even if the state of the node at that instance has changed since probing.

B. Receiver Initiated Policy

When a new job arrives at node S, the transfer policy would place the job either in the job queue or in the job transfer queue of node S as described before. The location policy is typically invoked by
nodes at times of job completions. The location policy of node S attempts to transfer a job from its job transfer queue to its job queue if the job transfer queue is not empty. Otherwise, if the job queue length of node S is less than T, it initiates the probing process as in the sender-initiated policy to locate a node D with a non-empty job transfer queue. If such a node is found within Pl probes, a job from the job transfer queue of node D will be transferred to the job queue of node S.

In this paper, as in the previous literature [5,18], we assume that T = 1. That is, load distribution is attempted only when a node is idle (Livny and Melman [14] call it 'poll when idle' policy). The motivation is that a node is better off avoiding load distribution when there is work to do.
Furthermore, several studies have shown that a large percentage (up to 80% depending on time of day) of workstations is idle [10, 13, 16, 19]. Thus the probability of finding an idle workstation is high.

Previous implementations of this policy have assumed that, if all probes fail to locate a suitable node to get work from, the node waits for the arrival of a local job. Thus, job transfers are initiated at most once every time a job is completed. This causes performance problems because the processing power is wasted until the arrival of a new job locally. This poses severe performance problems if the load is not homogeneous (e.g. if only a few nodes are generating the system load) [18] or if there is a high variance in job inter-arrival times [1]. For example, if four jobs arrive at a node in quick succession, then this node attempts load distribution only once after completing all four jobs. Worse still is the fact that if there is a long gap in job arrivals, the frequency of load distribution will be low. This adverse impact on performance can be remedied by reinitiating load distribution after the elapse of a predetermined time if the node is still idle. The receiver-initiated policy implemented here uses this reinitiating strategy.

C. Single Coordinator Policy

In this policy, there is a single node (called the "coordinator") that is responsible for collecting the system state information.

Whenever that state of a node changes, it informs this change in state to the coordinator for updating purposes. A node can be in one of three states:

- ❖ it is in a *receiver* state if the job queue length of the node is less than Tl (low threshold);
- ❖ it is in a *sender* state if the job queue length of the node is greater than Th (high threshold);

- Otherwise, it is in an OK state.

where Th ³ Tl. In this paper, for reasons explained in Section 2.2, we assume that Th = Tl = T =1.

The load distribution is initiated by a receiver node (i.e., a node that is in the receiver state). Typically, at job completion times, if the state of the node changes to receiver, it consults the coordinator node for a node that is in the sender state. If a sender node is found, the coordinator informs the sender node to transfer a job to the receiver node. As in the receiver-initiated policy, reinitiation of load distribution is necessary in order to improve its performance under certain system and workload conditions.

D. Hierarchical Policy

In the hierarchical policy, instead of a single node maintaining the entire system state, a set of nodes is given this responsibility. The system is logically divided into clusters and each cluster of nodes will have a single node that maintains the state information of the nodes within the cluster. The state information on the whole system is maintained in the form of a tree where each tree node maintains the state information on the set of processor nodes in the sub-tree rooted by the tree node. Figure 1 shows an example hierarchical organization for 8 processor nodes with a branching factor *B* of 2. Recall that a node can be in one of three states: *sender* (overloaded), *OK* (normal load), or *receiver* (underloaded). We will use +1 to represent the sender state, 0 for the OK state, and -1 for the receiver state. For example, Q4 maintains the state information in nodes N0 ands N1.

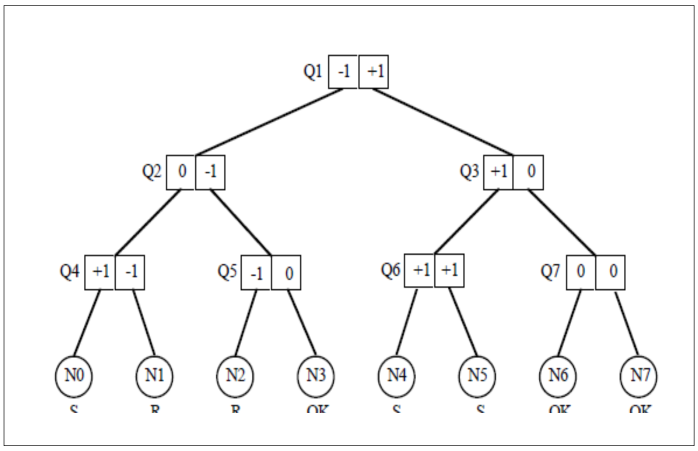

Fig. 1. An Example of Hierarchical Policy Tree with 8 nodes and branching factor 2 *(S-Sender node, R-Receiver node ,OK-OK node)*

It may be noted from Figure 3.1 that each node in the tree maintains the state information on all the nodes in the sub-tree rooted at this tree node. In other words, cluster size increases as we move up the tree. For example, Q2 maintains state information on nodes N0, N1, N2, and N3. However, in order to reduce the amount of information that has to be kept at higher tree nodes, only summary state information is maintained. The summary metric used in this policy is the arithmetic sum of the state information of the tree nodes below it. For example, summary state metric for Q4 is zero, which implies that the cluster represented by Q4 (i.e., nodes N0 and N1) is in OK state. Thus this policy encourages local load balancing. For this reason, this policy is called the *local hierarchical* policy. Also note that the summary metric for Q6 stored in Q3 is +1 rather than +2. This is because the value stored in the parent node is +1 if the sum is positive and -1 if it is zero or negative. An advantage of this scheme is that it reduces the number of updates required to maintain the hierarchy. For example, if the system state changes and N4 moves to OK state, only the entry in Q6 needs to be changed. Since N5 is still a sender, no state change is necessary for Q3. Dandamudi and Lo [2] discuss the impact of maintaining a true summary metric. In addition, they also discuss two global hierarchical policies.

III. Comparison of Proposed Model With Existing Methods

In this paper, we propose an algorithm by making use of hierarchical and single coordinator policy of load sharing techniques for heterogeneous distributed system application. In this locally distributed system model, There are 32 nodes taken as a collection in communication network at a higher level.

We model communication delays without modelling the low-level protocol details. An Ethernet-like network with 10 Mbits/sec is assumed. The communication network is modelled as a single server. Each node is assumed to have a communication processor that is responsible for handling communication with other nodes. Similar assumptions are made by other researchers [15]. The CPU would give preemptive priority to communication activities (such as reading a message, initiating the communication processor to send a probe message etc.) over the processing of jobs.

The CPU overheads to send/receive a probe and to transfer a job are modelled by *Tprobe* and *Tjx*, respectively. Actual job transmission (handled by the communication processor) time is assumed to be uniformly distributed between *Ujx* and *Ljx*.

Probing is assumed to be done serially, not in parallel. For example, the implementation [3] uses serial probing.

The system workload is represented by four parameters. The job arrival process at each class *i* node is characterized by a mean inter-arrival time 1/li and a coefficient of variation *C*ai. Jobs are characterized by a processor service demand on a class *i* node (with mean 1/mi) and a coefficient of variation *C*si. We study the performance sensitivity to variance in both inter-arrival times and service times (the CV values are varied from 0 to 4). We use a two-stage hyper exponential model to generate service time and inter arrival time CVs greater than 1 [8].

As in [15], there are two types of heterogeneous systems.

1. **Type-I Systems:** The nodes are homogeneous with same processing rate and job arrival rates at nodes are different.
2. **Type-II Systems:** the node processing rates as well as the job arrival rate at nodes are different.

To reduce the complexity of the experiments, we consider two node classes as in [15].

We consider two node classes in type II systems as well. Nodes in each node class in type II system can have different job arrival rates and service rates. We use the mean response time as the chief performance metric to compare the performance of the various load sharing policies.

A. Performance of Type-I Systems

The performance or simulation results for type I heterogeneous systems. Unless otherwise stated, the following default parameter values are assumed. The distributed system has *N* = 32 nodes interconnected by a 10 megabit/second communication network. The number of class 1 nodes *N*1 is 6 and the number of class 2 nodes *N*2 is 26.

The impact of node distribution between class 1 and class 2 is done with job service time. The average job service time is one time unit (e.g., 1 second) for both classes. That is m1 = m2 = 1. The size of a probe message is 16 bytes. The CPU overhead to send/receive a probe message *Tprobe* is 0.003 time units and to transfer a job *Tjx* is 0.02 time units.

The load distribution re initiation period when a “no job” (“false sender”) message is received is fixed at 1 (0.2). Job transfer communication overhead is uniformly distributed between *Ljx* = 0.009 and *Ujx* = 0.011 time units (i.e., average job transfer communication overhead is 1% of the average job service time).

Since we consider only non-executing jobs for transfer, 1% is a reasonable value. The threshold *Ts* (for the sender-initiated policies) is 2 and *TR* (for the receiver-initiated policies) is 1 (as explained in Section 2.1). Similar threshold values are used in previous studies [3-5]. Probe limit *Pl* is 3 for the sender-initiated and receiver-initiated policies.

Batch strategy has been used to compute confidence intervals (at least 30 batch runs were used for the results reported here).

This strategy produced 95% confidence intervals that were less than 1% of the mean response times when the system utilization is low to moderate and less than 5% for moderate to high system utilization (in most cases, up to about 90%).

B. Performance of Type-II Systems

In type II heterogeneous systems, nodes may have different processing rates in addition to seeing different job arrival rates as in type I systems. As in the type I systems, we consider a system with *N* = 32 nodes consisting of two node classes with processing rates of m1 = 0.5 and m2 = 1. Due to space limitations, we will only present a sample of the simulation results here.

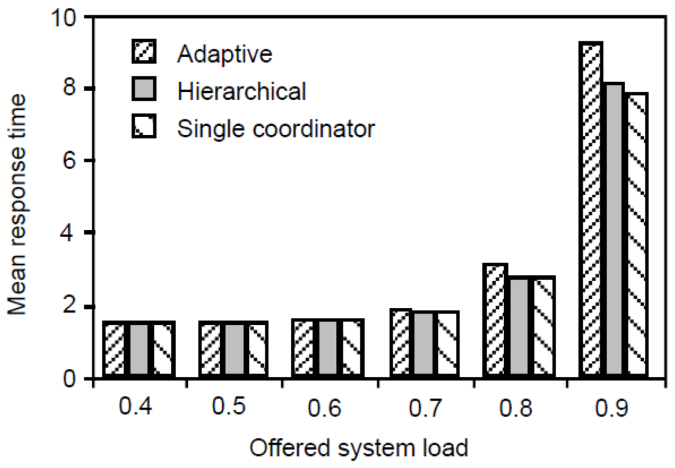

Fig. 2. Performance sensitivity to offered system load ($N = 32$ nodes, $N_1 = N_2 = 16$, $\lambda_1 = \lambda_2/2$, λ_2 is varied, $C_{a1} = C_{a2} = 4$, $\mu_1 = 0.5$, $\mu_2 = 1$, $C_{s1} = C_{s2} = 4$, $B = 8$, $T_l = T_h = 1$ for both classes, $P_l = 3$, transfer cost = 1%)

Figure 2 presents the results as a function of offered system load. For this experiment, we have divided the 32 system nodes equally between the two node classes (i.e., *N*1 = *N*2 = 16). The offered system load is maintained the same for both classes. That is, the arrival rate of class 1 is maintained at half of that for class 2 nodes because class 2 nodes are twice as fast (i.e., l1= l2/2). The threshold values *T*l and *T*h are fixed at 1 for both classes. The inter-arrival and service time CVs for both classes are fixed at 4.

All policies exhibit similar behavior as in Figure 1. The mean response times in Figure 4 are higher than those in Figure 2 as the system is operating at a higher average system load. The performance of the adaptive and hierarchical policies is worse by 14% and 1%, respectively, when the offered system load is 80%; the corresponding values are 17.8% and 3.8%, respectively, when the system load is increased to 90%. The reduced performance sensitivity is due to the fact that each class has 16 nodes as opposed 6 and 26 in type I experiment.

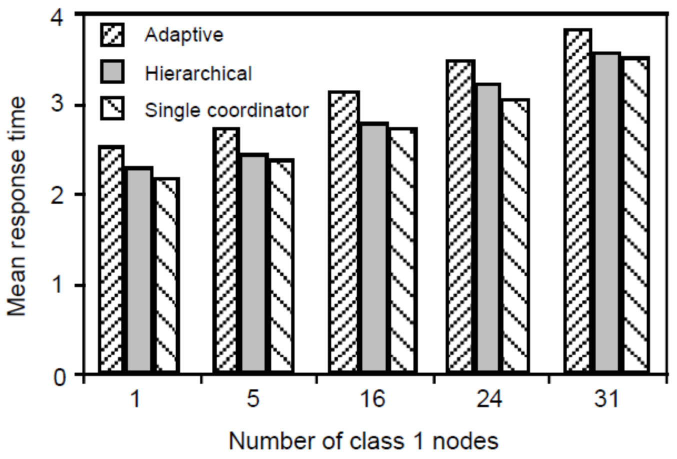

Fig. 3. Performance sensitivity to degree of heterogeneity ($N = 32$ nodes, $N1$ is varied, $N2 = N - N1$, $\lambda 1 = 0.4$, $\lambda 2 = 0.8$, Ca1 = Ca2 = 4, $\mu 1 = 0.5$, $\mu 2 = 1$, CS1 = CS2 = 4, $B = 8$, $T1 = Th = 1$ for both classes, $Pl = 3$, transfer cost = 1%)

We next look at the impact of degree of heterogeneity on the system performance. Figure 3 shows the response time as a function of number of class 1 nodes *N*1. Note that the number of class 2 nodes (i.e., the faster nodes) *N*2 is given by 32-*N*1. Thus as we move from left to right in Figure 6, the number of slower nodes increases in the system. As a result all policies experience an increase in response time. However, because each class of nodes is operating at the same system load of 80%, the performance sensitivity is much smaller than that shown in Figure 3 for the type I workload. Except for this difference, the policies exhibit similar behaviour as in Figure 3

IV. Conclusion

The centralized single coordinator policy is the best policy from the performance point of view in the absence of contention for the coordinator node. However, for large systems the coordinator may become a bottleneck limiting the performance benefits of such a policy. In addition, the single coordinator causes fault-tolerance problems as load sharing is dependent on this single coordinator node. Furthermore, in large hierarchically distributed networks (e.g., several LAN clusters connected by a WAN), consulting the central coordinator is expensive and leads to performance problems.

The hierarchical policy minimizes these performance problems. We have compared the performance of the hierarchical load sharing policy with that of the two distributed policies and the centralized single coordinator policy.

In order to see how close the hierarchical policy performs in comparison to the single coordinator policy, we have considered the scenario where the bottleneck problem does not exist in the centralized policy or single coordinator policy. It means that hierarchical policy performs very similar to the single coordinator policy (which provides the best performance in the absence of contention) for all the various systems.

Acknowledgment

The preferred spelling of the word “acknowledgment” in America….

References

[1] S. P. Dandamudi and M. Lo, *Hierarchical Load Sharing Policies for Distributed Systems*, Technical Report SCS- 96-1, School of Computer Science, Carleton University, Ottawa, Canada.

[2] P. Dikshit, S. K. Tripathi, and P. Jalote, “SAHAYOG: A Test Bed for Evaluating Dynamic Load-Sharing Policies,” *Software - Practice and Experience,* Vol. 19, No. 5, May 1989, pp. 411-435.

[3] D. L. Eager, E. D. Lazowska, and J. Zahorjan, "Adaptive Load Sharing in Homogeneous Distributed Systems," *IEEE Trans. Software Engng.*, Vol. SE-12, No. 5, May 1986, pp. 662-675.

[4] D. L. Eager, E. D. Lazowska, and J. Zahorjan, "A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing," *Performance Evaluation*, Vol. 6, March 1986, pp. 53-68.

[5] D. L. Eager, E. D. Lazowska, and J. Zahorjan, "The Limited Performance Benefits of Migrating Active Processes for Load Sharing," *ACM Sigmetrics Conf.*, 1988, pp. 63-72.

[6] A. Hac and X. Jin, “Dynamic Load Balancing in a Distributed System Using a Decentralized Algorithm,” *IEEE Int. Conf. Dist. Computing Systems,* 1987, pp. 170- 177.

[7] H. Kobayashi, *Modeling and Analysis: An Introduction to System Performance Evaluation Methodology,* Addison-Wesley, Reading 1981.

[8] P. Krueger and M. Livny, “A Comparison of Preemptive and Non-Preemptive Load Distributing,” *IEEE Int. Conf. Dist. Computing Systems,* 1988, pp. 123-130.

[9] P. Krueger and R. Chawla, “The Stealth Distributed Scheduler,” *IEEE Int. Conf. Dist. Computing Systems,* 1991, pp. 336-343.

[10] T. Kunz, “The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme,” *IEEE Trans. Software Engng.*, Vol. SE-17, No. 7, July 1991, pp. 725-730.

[11] W. E. Leland and T. J. Ott, “Load Balancing Heuristics and Process Behavior,” *Proc. PERFORMANCE 86* and *ACM SIGMETRICS 86*, 1986, pp. 54-69.

[12] M. J. Litzkow, M. Livny, and M. W. Mutka, “Condor – A Hunter of Idle Workstations,” *IEEE Int. Conf. Dist. Computing Systems,* 1988, pp. 104-111.

[13] S. P. Dandamudi, “Performance Impact of Scheduling Discipline on Adaptive Load Sharing in Homogeneous Distributed Systems,” *IEEE Int. Conf. Dist. Computing Systems,* 1995, pp. 484-492.

[14] M. Livny and M. Melman, “Load Balancing in Homogeneous Broadcast Distributed Systems, *Proc. ACM Computer Network Performance Symp*., 1982, pp. 47-55.

[15] R. Mirchandaney, D, Towsley, and J. A. Stankovic, “Adaptive Load Sharing in Heterogeneous Distributed Systems,” *J. Parallel and Distributed Computing,* Vol. 9, 1990, pp. 331-346.

[16] M. Mutka and M. Livny, “Profiling Workstation’s Available Capacity for remote Execution,” *Proc Performance 87,* Brussels, Belgium, 1987, pp. 529-544.

[17] N. G. Shivaratri and P. Krueger, “Two Adaptive Location Policies for Global Scheduling Algorithms,” *IEEE Int.Conf. Dist. Computing Systems,* 1990, pp. 502-509.

[18] N. G. Shivaratri, P. Krueger, and M. Singhal, “Load Distributing for Locally Distributed Systems,” *IEEE Computer,* December 1992, pp. 33-44.

[19] M.M. Theimer and K. A. Lantz, “Finding Idle Machines in a Workstation-Based Distributed System,” *IEEE Int. Conf. Dist. Computing Systems,* 1988, pp. 112-122.

[20] Y. T. Wang and R. J. T. Morris, “Load Sharing in Distributed Systems,” *IEEE Trans, Computers,* Vol. C-34, March 1985, pp. 204-217.

[21] S. Zhou, “A Trace-Driven Simulation Study of Dynamic Load Balancing,” *IEEE Trans. Software Engng.*, Vol. SE- 14, No. 9, September 1988, pp. 1327-1341.

[22] S. Zhou, X. Zheng, J. Wang, and P. Delisle, “Utopia: a Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems,” *Software - Practice and Experience,* Vol. 23, No. 12, December 1993, pp. 1305-1336.

D.Sharanya M.Tech, CSE Dept., Institute of Aeronautical Engineering,HYD-500043,TS,India.

Dr. N. Chandra Sekhar Reddy Professor, CSE Dept., Institute of Aeronautical Engineering, HYD-500043,TS, India.

Baswanth Professor, CSE Dept., Institute of Aeronautical Engineering, HYD-500043, TS, India.