Load Balancing In Distributed Computing

Pranit H Bari, Student, Department of Computer Engineering and Information Technology, VJTI, Mumbai
B B Meshram, HOD, Department of Computer Engineering and Information Technology, VJTI, Mumbai

ABSTRACT
The paper presents what a Distributed system is. It also overviews the issues to be taken into consideration for developing the strategies for load balancing. Strategies for various types of Distributed systems are discussed. Each strategy considers the most economical transfer policy, selection policy, location policy, information policy. The paper concludes with comparison of various strategies for various types of Distributed systems. Finally the paper presents some of the economical strategies for Distributed Systems.
Keywords: Distributed systems, Network systems, task migration, load sharing, task scheduling, process queue

Introduction
A Distributed system is a collection of processor-memory pairs connected by a communications subnet and logically integrated in varying degrees by a distributed operating system and/or distributed database system. The increased demand for Distributed system is due to ever rising need for communications, development in network systems and cost effective resource sharing softwares, economical sharing of resources and productivity.
A Distributed system provides various advantages good resource sharing, reliability, extensibility, good performance. Potential reliability improvements are due to the data and control redundancy possible, the geographical distribution of the system, and the ability for mutual inspection of hosts and communication processors. Extensibility is the ability to easily adapt to both short and long term changes without significant disruption of the system.

Issues In Distributed System
This section provides a perspective on six interrelated Distributed System issues: the object model, access control, heterogeneity, openness, failure handling, concurrency, and transparency.

1. **Object Model**
All hardware and software resources can be regarded as objects. All resources required at some instance are referred to as domain. The communication between 2 or more hosts can be synchronous or asynchronous.

2. **Access Control**
Access control refers to the manner in which the objects are accessed depending on the requirement of the system and the usage of resources by a restricted set of users. Access control can be implemented by the usage of serial locking like polling, token sharing, locking.

3. **Heterogeneity**
Heterogeneity can be applied to networks, computer hardware, operating systems, programing languages, implementations by various developers. The solution is translation at the sender host or at the receiver host by the distributed system. There are two different solutions applicable under different circumstances: an intermediate translator or an intermediate standard data format.

4. **Openness**
The openness of the system is the characteristic that determine whether the system can be extended and re-implemented in various ways. The openness of a distributed system can be determined by the degree to which new resource sharing services can be added and made available

5. **Failure Handling**
Failures in distributed system are partial, i.e. some components fail while others continue to run. A distributed system should be able to detect failures, mask them from the users and should be able to tolerate them. Recovery of a system can be done by using the concept of 'roll back' in case of a crash. Services can be made to tolerate failures by the use of redundant components.

6. **Concurrency**
There is a possibility that several clients may attempt to access a particular shared resource simultaneously. Such incidents may cause data conflicts and thus result in data inconsistency. Such inconsistency can be avoided by the usage of 'semaphores' which is just a lock and access based technique used to avoid mutual exclusion.

Taxonomy Of Scheduling

We may view every instance of the scheduling problem as consisting of three main components.
1) Consumer(s).
2) Resource(s).
3) Policy.

One can observe the behavior of the scheduler in terms of how the policy affects the resources and consumers. This relationship between the scheduler, policies, consumers, and resources is shown in Fig. 1.

The Classification Scheme:

1. Hierarchical Classification:

a) Local Versus Global:
At the highest level, we may distinguish between local and global scheduling. Local scheduling is involved with the assignment of processes to the time-slices of a single processor.

b) **Static versus Dynamic:**
Static scheduling may include algorithms that schedule task forces for a particular hardware configuration. Over a period of time, the topology of the system may change, but characteristics describing the task force remain the same. Hence, the scheduler may generate a new assignment of processes to processors to serve as the schedule until the topology changes again.

c) **Optimal versus Suboptimal**
In the case that all information regarding the state of the system as well as the resource needs of a process are known, an optimal assignment can be made based on some criterion function. In the event that these problems are computationally infeasible, suboptimal solutions may be tried.

d) **Approximate versus Heuristic**
Approximate is to use the same formal computational model for the algorithm,but instead of searching the entire solution space for an optimal solution, we are satisfied when we find a "good" one.
This branch represents the category of static algorithms which make the most realistic assumptions about a priori knowledge concerning process and system loading characteristics. It also represents the solutions to the static scheduling problem which require the most reasonable amount of time and other system resources to perform their function.

2. Flat Classification Characteristics:

a) Adaptive Versus Non-adaptive
An adaptive solution to the scheduling problem is one in which the algorithms and parameters used to implement the scheduling policy change dynamically according to the previous and current behavior of the system in response to previous decisions made by the scheduling system.

A non-adaptive scheduler would be one which does not necessarily modify its basic control mechanism on the basis of the history of system activity.

b) **Load Balancing:**
The basic idea is to attempt to balance (in some sense) the load on all processors in such a way as to allow progress by all processes on all nodes to proceed at approximately the same rate.

Normally, information would be passed about the network periodically or on demand in order to allow all nodes to obtain a local estimate concerning the global state of the system.

Then the nodes act together in order to remove work from heavily loaded nodes and place it at lightly loaded nodes.

c) Bidding:
The manager represents the task in need of a location to execute, and the contractor represents a node which is able to do work for other nodes.

The manager announces the existence of a task in need of execution by a task announcement, and then receives bids from the other nodes (contractors).

Strategies

1. For Locally Distributed System:

Components for Locally Distributed System Load Balancing are:

a) **Transfer policy**
A transfer policy determines whether a node is in a suitable state to participate in a task transfer, either as a sender or a receiver. Many proposed transfer policies are threshold policies. Thresholds are expressed in units of load.

When a new task originates at a node, the transfer policy decides that the node is a sender if the load at that node exceeds a threshold TI.

b) **Selection policy**
Once the transfer policy decides that a node is a sender, a selection policy selects a task for transfer. Should the selection policy fail to find a suitable task to transfer, the node is no longer considered a sender. The simplest approach is to select one of the newly originated tasks that caused the node to become a sender. A selection policy considers several factors in selecting a task:
The overhead incurred by the transfer should be minimal. For example, a small task carries less overhead.
The selected task should be long lived so that it is worthwhile to incur the transfer overhead.
The number of location-dependent system calls made by the selected task should be minimal. Location-dependent calls are system calls that must be executed on the node where the task originated, because they use resources such as windows, the clock, or the mouse that are only at that node.

c) **Location policy**
The location policy's responsibility is to find a suitable "transfer partner" (sender or receiver) for a node, once the transfer policy has decided that the node is a sender or receiver. A widely used decentralized policy finds a suitable node through polling. In a centralized policy, a node contacts one specified node called a coordinator to locate a suitable node for load sharing. The coordinator collects information about the system (which is the responsibility of the information policy), and the transfer policy uses this information at the coordinator to select receivers.

d) **Information policy**
The information policy decides when information about the states of other nodes in the system is to be collected, from where it is to be collected, and what information is collected. There are three types of information policies:

i. Demand-driven policies
ii. Periodic policies.
iii. State-change-driven policies.

Based on these components there are 4 strategies namely:

1. Sender-initiated Algorithm:

a) **Transfer policy:** A node is identified as a sender if a new task originating at the node makes the queue length exceed a threshold T.

b) **Selection policy:** All three algorithms have the same selection policy, considering only newly arrived tasks for transfer.

c) **Location policy:**
i. Random
ii. Threshold
iii. Shortest

d) **Information policy**: When either the shortest or the threshold location policy is used, polling starts when the transfer policy identifies a node as the sender of a task. Hence, the information policy is demand driven.

2. **Receiver-initiated algorithm:**

a) **Transfer policy**
The algorithm's threshold transfer policy bases its decision on the CPU queue length. If the local queue length falls below the threshold T, then the node is identified as a receiver for obtaining a task from a node (sender) to be determined by the location policy

b) **Selection policy**
The algorithm considers all tasks for load distributing, and can use any of the approaches under "Selection policy"

c) **Location policy**
Selects a node at random and polls it to determine whether transferring a task would place its queue length below the threshold level. If not, then the polled node transfers a task.

d) **Information policy**
Demand driven, since polling starts only after a node becomes a receiver.

3. Symmetrically-initiated Algorithm:

Both senders and receivers initiate load-distributing activities for task transfers. At low system loads, the sender-initiated component is more successful at finding underloaded nodes. At high system loads, the receiver-initiated component is more successful at finding overloaded nodes.

4. Adaptive algorithm:

The stable symmetrically initiated algorithm' uses the information gathered during polling (instead of discarding it, as the previous algorithms do) to classify the nodes in the system as sender overloaded, receiver underloaded, or OK (nodes having manageable load). The knowledge about the state of

nodes is maintained at each node by a data structure composed of a senders list, a receivers list, and an OK list.

Dynamic Load Balancing On Highly Parallel Computers:

1. The Gradient Model (GM)

The gradient model is a demand driven approach. The basic concept is that underloaded processors inform other processors in the system of their state, and overloaded processors respond by sending a portion of their load to the nearest lightly loaded processor in the system. The scheme is based on two threshold parameters: the Low-Water-Mark (LWM) and the High- Water-Mark (HWM). A processor's state is considered light if its load is below the LWM, heavy if above the HWM, and moderate otherwise. A node's proximity is defined as the shortest distance from itself to the nearest lightly loaded node in the system. All nodes are initialized with proximity of Wmax, constant equal to the diameter of the system. The proximity of a node is set to 0 if its state becomes light. In order for load balancing to take place, there must be at least one overloaded processor and one underloaded processor in the system.

$$Lp - Lq > HWM - LWM$$

The proximity map is used to perform the migration phase. If a processor's state is heavy and any of its near-neighbors report a proximity less than Wmax, then it sends a unit of its load to the neighbor of lowest proximity. Tasks are routed through the system in the direction of the nearest underloaded processors.

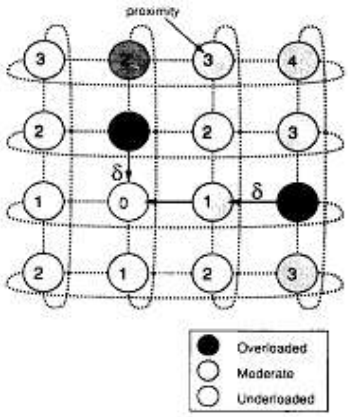

Fig. The Gradient Model

2. Sender Initiated Diffusion (SID)

The SID strategy is a, local, near-neighbor diffusion approach which employs overlapping balancing domains to achieve global balancing. The scheme is purely distributed and asynchronous. Each processor acts independently, apportioning excess load to deficient neighbors.For an N processor system with a total system load L unevenly distributed across the system, a diffusion approach, such as the SID strategy, will eventually cause each processor's load to converge to L/N. Balancing is performed by each processor whenever it receives a load update message from a neighbor indicating that the neighbors load, Li, < Llow, where Llow is a preset threshold. All processors inform their near neighbors of their load levels and update this information throughout program execution.Task migration is performed by apportioning excess load to deficient neighbors. Each neighbor k is assigned a weight hk according to the following formula,

Hk=Lp-lk iff lk<Lp
=0 otherwise

Finally, the portion of processor p's excess load that is assigned to neighbor k,Sk , is defined as

Sk=(lp-Lk)hk\Hp

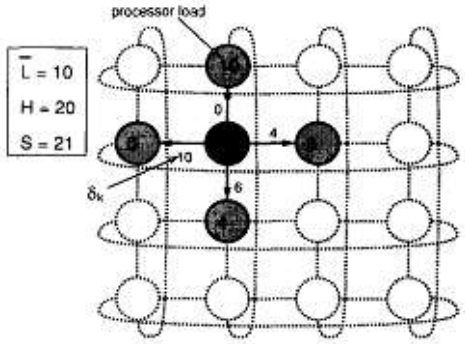

Fig. The SID Strategy

3. Receiver Initiated Diffusion (RlD):

It is a receiver initiated approach in the RID strategy underloaded processors request load from overloaded neighbors. First, the balancing process is initiated by any processor whose load drops below a pre-specified threshold (LLow) . Second, upon receipt of a load request, a processor will fulfill the request only up to an amount equal to half of its current load (this reduces the effect of the aging of the data upon which the request was based). Finally, in the receiver initiated approach the underloaded processors in the

system take on the majority of the load balancing overhead, which can be significant when the task granularity is fine. When a processor's load drops below the prespecified threshold LLOW, the profitability of load balancing is determined by first computing the average load in the domain,Lp . If a processor's load is below the average load by more than prespecified amount,Lthreshold, it proceeds to implement the third phase of the load balancing process,

$$\bar{L}_p - L_p > L_{threshold}.$$

Task migration is performed by requesting proportionate amounts of load from overloaded neighbors. Each neighbor k is assigned a weight h k according to the following formula,

$$h_k = \begin{cases} l_k - \bar{L}_p, & \text{if } l_k > \bar{L}_p, \\ 0 & \text{otherwise.} \end{cases}$$

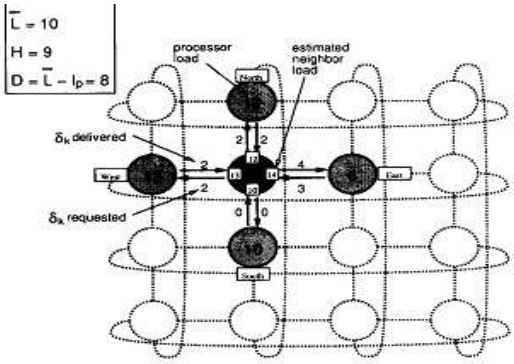

Fig. The RID Strategy

4. Hierarchical Balancing Method (HBM):

The HBM strategy organizes the multicomputer system into a hierarchy of balancing domains, thereby decentralizing the balancing process. Specific processors are designated to control the balancing operations at different levels of the hierarchy. In this case, processors in charge of the balancing process at a level, li, receive load information from both lower level, l z - l , domains. Global balancing is achieved by ascending the tree and balancing the load between adjacent domains at each level in the hierarchy.

Subtree load information is computed at intermediate nodes and propagated to the root. The update policy for intermediate nodes is the same as that of leaf processors. Load imbalances at different levels of the hierarchy are detected at intermediate nodes. If the load imbalance between two lower level domains is greater than a prespecified amount, Lthreshold, then one domain is considered overloaded and the other underloaded.The absolute value of the difference between the left domain load, LL, and the right domain load, LR, is compared to Lthreshold.

Ll-Lr>Lthreshold

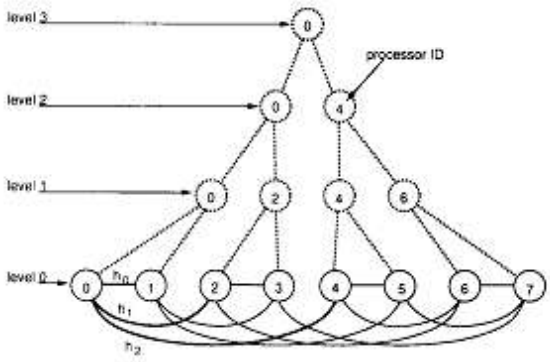

Fig. The HBM Strategy

Task migration is controlled by intermediate nodes which, upon detecting an imbalance, notify all processors belonging to the overloaded subtree of the amount of the imbalance and at what level it occurs. Processors within the overloaded branch transfer a designated amount of their load to their "matching" neighbor in the adjacent underloaded subtree. For each processor in the left subtree at a given level, there is a corresponding processor in the right subtree, and visa-versa.

5. The Dimension Exchange Method (DEM):

The DEM strategy was conceptually designed for a hypercube system but may be applied to other topologies with some modifications. In the case of an N processor hypercube configuration, balancing is performed iteratively in each of the logN dimensions. All processor pairs in the first dimension, those processors whose addresses differ in only the least significant bit, balance the load between themselves. Next, all processor pairs in the second dimension balance the load between themselves, and so forth, until each processor has balanced its load with each of its neighbors.

Balancing is initiated by any underloaded processor which has a load that drops below a preset threshold.

Lp>Lthreshold

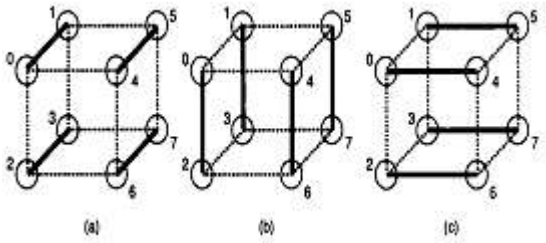

Fig. The DEM Strategy

This processor broadcasts a load balancing request to all other processors in the system.

Discussion

When considering performance from the consumer (or user) point of view, the metric involved is often one of minimizing individual program completion times-response. Alternately, the resource point of view also considers the rate of process execution in evaluating performance, but from the view of total system throughput. In contrast to response, throughput is concerned with seeing that all users are treated fairly and that all are making progress. Notice that the resource view of maximizing resource utilization is compatible with the desire for maximum system throughput.

1. **For Locally distributed System**:

FIG shows that the sender-initiated algorithm with a threshold location policy performs marginally better than the receiver-initiated algorithm at light to moderate system loads, while the receiver-initiated algorithm performs substantially better at high system loads (even though the preemptive transfers it uses are much more expensive than the nonpreemptive transfers used by the sender-initiated algorithm).

For symmetrically initiated load sharing this policy takes advantage of its sender- initiated load-sharing component a low system loads, its receiver-initiated component at high system loads, and both at moderate system loads. Hence, its performance is better than or matches that of the sender-initiated algorithm with threshold location policy at all levels of system load, and is better than that of the receiver-initiated policy at low to moderate system loads.

2. **For Dynamic Load Balancing**:

GM complexity :The overhead incurred by the GM strategy includes the updating of the proximity map and the routing of tasks from overloaded processors to underloaded ones. The number of messages required to update the proximity map is dependent on the interconnection network topology, the number of underloaded processors, their locations, and the order in which they become underloaded.

3. **SID Complexity**:

The SID near-neighbor balancing scheme distributes the load balancing overhead uniformly across all processors. For a K-connected (K neighbors) system of N processors, a complete system load update can be expressed in terms of the number of messages sent,

Ctot(update) = KN messages

C(update)=KN/K=K messages

TABLE I
SUMMARY OF COMPARISON ANALYSIS

Category	GM	SID	RID	HBM	DEM
Initiation	receiver	sender	receiver	designated	designated
Balancing Domain	variable	overlapped	overlapped	variable	variable
Knowledge	global*	local	local	global	global
Aging Period	$O(\text{diameter}(N))$	$f(u, K)$	$f(u, K)$	$f(u, N)$	constant
Overhead Distribution	uniform†	uniform	uniform	nonuniform	uniform

Recall, N is the number of processors, u is the update factor, and K is the number of near-neighbors.
* global knowledge of the locations of underloaded processors, but limited knowledge concerning the quantity of the imbalance.
† statistically uniform for a random load distribution.

FIG.2 Comparison For Dynamic Load Balancing

4. RID Complexity:

The RID near-neighbor balancing scheme distributes the load balancing overhead uniformly across all processors. For a K-connected (K neighbors) system of N processors, a complete system load update can be expressed in terms of the number of messages sent,

Ctot(update) = KN messages

And the overhead incurred is

C(update)=KN/K=K messages

5. **Hierarchial Complexity**

For a system of N processors with an embedded binary-tree communication structure (Fig. 4), a full system load update from the leaves to the root requires N - 1 update messages that may be completed in log N stages.

6. **DEM Complexity**

The DEM strategy overhead is distributed uniformly across all processors in the system. Once synchronized, each processor, in an N processor system, is involved in logN balancing operations. Each of these operations includes a load update message, a load transfer request message, and a load transfer (which may require multiple essages). Each processor will incur 2 sends and 1 receive or 1 send and 2 receives depending on whether or not they are designated to compute the imbalance between the pair. In any case, the balancing process is evenly distributed with a total communication overhead, Ctot,

Ctot = 3N log N(messages).

Proposed System

The centralized as well as distributed system, both have their disadvantages .however if we consider their advantages as standalone they might both together be efficient for the functioning of load balancing. Here are some of the factors why we need to develop a combining system of both distributed as well as centralized system:

1. Since each node in the system makes autonomous decisions, no node can envision the precise state of the whole system. Thus a node has to make either a probabilistic decision or some kind of guess about the system state.
2. A task after going through communication set-up times, queueing, and transmission delays over multiple links may find that the destination node that was originally conceived of as the best choice has become highly loaded due to arrivals from some other nodes
3. Gathering a large amount of state information may decrease the accuracy,
4. The control overhead depends on the system load and can be high at heavy loading conditions.
5. Message exchange overhead linearly increases with the system size. This may result in a proportional increase in the average response time and extra communication traffic that can impede regular communication activity.
6. Centralized algorithms require accumulation of global information which can become a formidable task
7. The storage requirement for maintaining the state information also becomes prohibitively high with a centralized model of a large system
8. For a large system consisting of a hundred or thousand nodes, the central scheduler can become a bottleneck and can lower the throughput.

Strategy:

The CDLB is a two level strategy. At the first level the load is balanced among different spheres of the system, thus providing a "distributed environment" among spheres. At the second level, load balancing is carried out within individual spheres where the scheduler of each sphere acts as a centralized controller for its own sphere. The design of such a strategy involves the following steps

1. Design a network diagram for the entire system
2. Select the nodes which can become controllers
3. State an algorithm for load balancing within the controller as well as among the controllers
4. Strategy for collecting load information and maintaining state information

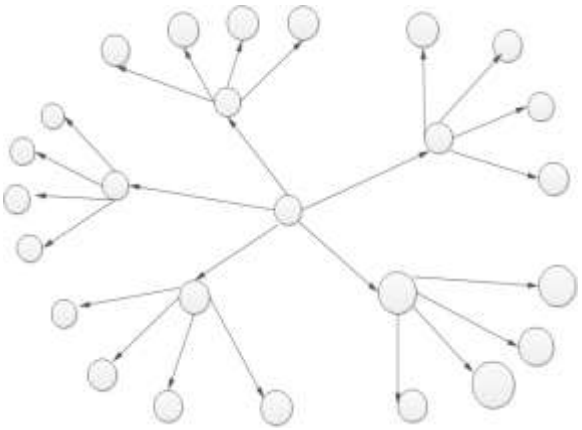

Fig: CDLB network diagram

The CDLB system maintains doubly ended queue at each scheduler. The queues are arranged in the order of the decreasing load each node contains. The queue has 2 pointers-top and bottom. Furthermore the scheduler first inspects whether the request is from sender or receiver. Each node is assigned with an address that corresponds to the sphere it belongs to. The sphere has a predefined threshold value that indicates whether load at a particular node is overloaded or underloaded.

Algorithm

1. At Node

Calculate load
If load>threshold1 send request to scheduler
If load<threshold2 send request to scheduler

2. At Scheduler

STEP 1: Check whether request is from sender or receiver.
STEP 2:for receiver:
2.1: go to queue[top]
2.2: if threshold1<queue[0] transfer load to that node that sent request.
update queue and return node address
else
send task to next hierarchy
Repeat step 2 at that scheduler
Update sphere[load]
Step 3: for sender:
3.1 go to queue[bottom]
3.2 if threshold2>queue[0] transfer load to that node that sent request.
Update queue and return node address
1.3 else
send request to next hierarchy
Repeat step 3 at that scheduler
STEP 4:STOP

Conclusion

We will not achieve the full potential advantages of DCS's until better experimental evidence is obtained and until current and new solutions are better integrated in a systemwide, flexible, and cost effective manner. Major breakthroughs will be required to achieve this potential. The heuristics, though, will have to directly address the problems of distribution, such as long delays, the assignment of credit, missing and out-of date information, the use of statistical information, and failure events. To achieve the objectives of DCS's, it is important to study distributed resource management paradigms
that view resources at an integrated "system" level
which includes the hardware, the communication subnet, the operating system, the database system, the programming language, and other software resources.

In case of Dynamic Load Balancing two major issues, that of load balancing overhead and the degree of knowledge used in balancing decisions are considered.. Of the five strategies proposed, the DEM strategy tended to outperform the rest for all granularities. The efficiency of the DEM and the HBM strategies, depends heavily on the system interconnection topology. The overhead of synchronization costs [scale as O(NlogN)] for the DEM approach and the aging period and nonuniform overhead distributions of the HBM approach may deteriorate their performance when the number of processors is large (1000 processors). The RID strategy, on the other hand, is easily ported to simpler topologies, and can scale gracefully for larger systems. Finally, for a wider variety of applications, exhibiting local communication dependencies between tasks, the RID scheme is able to maintain task locality. Therefore, since its performance was shown to be comparable to those of the DEM and HBM approaches, the RID strategy may be best suited for a broader range of systems supporting a large variety of applications.

References

[1] John A. Stankovic, member, IEEE "A perspective on distributed computer systems" IEEE Transactions on Computers, vol. C-33, no. 12, December 1984.
[2] Thomas L. Casavant, member, IEEE, and Jon G. Kuhl, member, IEEE
[3] "A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems" IEEE Transactions on Software Engineering, vol. 14, no. 2, February 1988.
[4] Edmund H. Durfee, member IEEE, Victor R. Lesser, and Daniel D. Corkill, member IEEE "Trends in Cooperative Distributed problem solving" IEEE transactions on knowledge and data engineering, vol. I . No. I . March 1989.
[5] Niranjan G. Shivaratri , Phillip Krueger, and Mukesh Singhal Ohio State University "Load Distributing for Locallv Distributed Systems" December 1992.
[6] Ishfaq Ahmad, Student Member, IEEE, and Arif Ghafoor, Senior Member, IEEE "Semi-

Distributed Load Balancing For Massively Parallel Multicomputer systems" IEEE Transactions on Software Engineering, vol. 17, no. 10, October 1991.

[7] Tony T. Y. Suen and Johnny S. K. Wong "Efficient Task Migration Algorithm for Distributed Systems" IEEE Transactions on Parallel and Distributed Systems, vol. 3, no. 4, July 1992.

[8] Marc H. Willebeek-lemair, Member, IEEE, and Anthony P. Reeves, Senior Member, IEEE "Strategies for Dynamic Load Balancing On Highly Parallel computers" IEEE Transactions on parallel and Distributed Systems, vol. 4, no. 9, September 1993.

[9] Kumar K. Goswami, Murthy Devarakonda, and Ravishankar K. Iyer, Fellow, IEEE "Prediction-Based Dynamic Load-Sharing Heuristics" IEEE Transactions on Parallel and Distributed Systems, vol. 4, no. 6, June 1993.

[10] Ivor Page, Tom Jacob, and Eric Chern "Fast Algorithms for Distributed Resource Allocation" IEEE Transactions On Parallel And Distributed Systems, Vol. 4, No. 2, February 1993

[11] Mahadev Satyanarayanan, Member James J. Kistler, Puneet Kumar, Maria E. Okasaki, Ellen H. Siegel, And David C. Steere "Coda: A Highly Available File System For A Distributed Workstation Environment" IEEE Transactions On Computers, Vol. 39, No. 4, April 1990

[12] Douglas B. Terry, Member, Ieee "Caching Hints In Distributed Systems" Ieee Transactions On Software Engineering, Vol. Se-13, No. 1, January 1987

[13] W.A Watson 111, J. Chen, G. Heyes, E. Jastrzembski, D. Quarrie "Coda A Scalable, Distributed Data Acquisition System" IEEE Transactions on Nuclear Science,VOl. 41, No. 1. February 1994

[14] Daniel Grosu, Student Member, IEEE, And Anthony T. Chronopoulos, Senior Member, IEEE "Algorithmic Mechanism Design For Load Balancing In Distributed Systems" Ieee Transactions On Systems, Man, And Cybernetics—Part B: Cybernetics, Vol. 34, No. 1, February 2004

[15] Orly Kremien and Jeff Kramer "Methodical Analysis of Adaptive Load Sharing Algorithms" IEEE Transactions On Parallel And Distributed Systems, VOL. 3, NO. 6, NOVEMBER 1992

[16] Wei Shu, Member, IEEE, and Min-You Wu, Senior Member, IEEE "Runtime Incremental Parallel Scheduling (RIPS) on Distributed Memory Computers" IEEE Transactions On Parallel And Distributed Systems, Vol. 7, No. 6, JUNE 1996

[17] Kang G. Shin, Fellow, IEEE, and Chao-Ju Hou, Student Member, IEEE "Analytic Models of Adaptive Load Sharing Schemes in Distributed Real-Time Systems" IEEE Transactions On Parallel And Distributed Systems, Vol. 4, No. 7, July 1993

[18] Lionel M. Ni, Member, IEEE, Chong-Wei Xu, And Thomas B. Gendreau, Student Member, IEEE "A Distributed Drafting Algorithm for Load Balancing" IEEE Transactions On Software Engineering, Vol. Se-Il, No. 10, October 1985