LOAD BALANCING TECHNIQUES

Two important characteristics of distributed systems are resource multiplicity and system transparency. In a distributed system we have a number of resources interconnected by a communication network. Apart from information sharing another important use of distributed systems is sharing of the processing power among various nodes. This can be done by migrating a local process and executing it at a remote node of the network. One reason for this migration could be that the node on which the process is submitted is not capable of executing it.

Another reason could be that the local node has to be shut down for some reason, so the process has to be migrated somewhere else. A process may also be executed remotely if the expected turnaround time will be better. Hence we can say that a distributed system works as a single virtual system and a resource in this system could either be logical, such as a shared file, or physical, such as a CPU.

3.1: NEED FOR LOAD BALANCING

We know that there are a number of nodes in a distributed system. The tasks that arrive in these systems are typically not uniformly distributed. Some of the nodes may be heavily loaded while the others may be lightly loaded. It may be possible to increase the overall throughput of the system if we allow heavily loaded nodes to redistribute tasks to lightly loaded nodes. This is known as load balancing. The aim of load balancing is to try to improve the performance of a distributed system, mainly in terms of resource availability or response time by distributing workload amongst a set of cooperating hosts.
3.2: DEGREE OF LOAD DISTRIBUTION

One of the main objectives of load distribution is to divide the workload amongst a group of cooperating hosts. The amount by which this objective can be fulfilled depends upon the architecture of the system and the environment.

Load distribution is described in the literature in a variety of widely differing techniques and methodologies. These are broadly classified as:

- Task Assignment Approach
- Load Sharing
- Load Balancing
- Load Leveling

3.2.1: Task Assignment Approach

In this approach each process submitted by a user for processing is viewed as a collection of related tasks and these tasks are scheduled to suitable nodes so as to improve performance.

The task assignment problem is well-known to be NP-hard. In most cases a task-assignment algorithm seeks an assignment that optimizes a certain cost function—for example, maximum throughput or minimum turnaround time. However, most reported algorithms yield suboptimal solutions. In general, optimal solutions can be found through an exhaustive search, but because there are \( n^m \) ways in which \( m \) tasks can be assigned to \( n \) processors, an exhaustive search is often not possible. Thus, optimal-solution
algorithms exist only for restricted cases or very small problems. The other possibility is to use an informed search to reduce the state space.

Like other NP-hard problems, there are three common ways to tackle task assignment:

• **Relaxation.** You can relax some of the requirements or restrict the problem.

• **Enumerative optimization.** If you can’t compromise the solution’s optimality, you can use enumerative methods, such as dynamic programming and branch and-bound.

• **Approximate optimization.** You can use heuristics to solve the problem while aiming for a near-optimal or good solution.

There exist polynomial time exact algorithms for solving the task assignment problem (TAP) in two-processor distributed systems [28]. However, the general n-processor TAP has been found to be NP-complete [29]. Therefore, finding exact solutions to large-scaled TAP is computationally prohibitive. The existing approaches for tackling the TAP can be divided into three categories.

(1) Mathematical programming approaches [30-32] using column generation or branch-and-bound techniques can solve the problem more efficiently.

(2) Customized algorithms [33-37] have been developed for providing exact solutions in specific circumstances, such as the distributed systems with linear processor array, processor mesh, and partial k-tree communication graph.

(3) Meta-heuristic algorithms involving genetic algorithms (GA) [38-40] and simulated annealing (SA) [41, 42] have been used to derive approximate solutions with reasonable
time. The development of meta-heuristic optimization theory has been flourishing during the last decade. Many new search paradigms such as tabu search [43] and ant colony optimization [44] have shown their efficacy in solving computationally intensive problems. This suggests the exploration of the potentials for solving the TAP using new meta-heuristics. Recently, Kennedy and Eberhart [45] developed a new evolutionary algorithm called particle swarm optimization (PSO). Reference [46] presents a PSO based algorithm for conquering the TAP. A hill climbing heuristic is embedded in the PSO iteration to expedite the convergence.

3.2.2: Load Sharing

The coarsest form of load distribution is load sharing. In this form of distribution load can only be placed on idle hosts. That is the load on a node is viewed as a binary value either 0 (idle) or 1 (at least one task is present). Some of the common load sharing systems are Sprite [47], Butler [48] and V [49]. This scheme just prevents the nodes from being idle while some other nodes have more than two processes. These algorithms normally employ the simplest load estimation policy of counting the total number of processes on a node.

3.2.3: Load Balancing

Contrary to load sharing, load balancing is the finest form of load distribution. This method tries to ensure that the workload on each host is within some small range (or balance criterion) of the workload present on all the other nodes in the system.
3.2.4: Load Leveling

Load leveling is a load distribution method which lies between the two extremes of load balancing and load sharing. Instead of trying to get a strictly even distribution of load across all the nodes, or simply making use of the idle nodes, load leveling tries to avoid congestion on any one host. This term is most of the time used interchangeably with load balancing.

In practical situation it is difficult to distinguish between load leveling and load balancing schemes. As the definition says, a load balancing scheme has to continue to redistribute load until and unless it meets a balance criterion. There is one such scheme from Kara [50] which only considers the system balanced if the difference between the most and the least loaded hosts is within a limit. Another scheme is MOSIX [51], which could be considered to be a load balancing systems but in fact it is a load leveling scheme as the balancing phase occurs periodically.

The degree of load balancing required depends upon the architecture of the system. Suppose we have a large network of personal workstations. Here it would be very expensive to load balance because of overheads of load and state collection. But the detection and utilization of idle workstations is easier as shown by the Butler system [48] so we can go for load sharing in this case.

As we move from load sharing to load leveling to load balancing we are actually moving from a coarse distribution to a fine distribution of load.
3.3: LOAD BALANCING APPROACHES

Load balancing problem has been discussed in traditional distributed systems literature for more than two decades. Various algorithms, strategies and policies have been proposed, implemented and classified.

3.3.1: Load Balancing Algorithms

The algorithms for load balancing can be classified into two categories: static or dynamic.

(a) Static Load Balancing Algorithm

Static load balancing algorithms allocate the tasks of a parallel program to workstations based on either the load at the time nodes are allocated to some task, or based on an average load of our workstation cluster. Previous research on static and dynamic load balancing can be found in [52]-[56]. The decisions related to load balance are made at compile time when resource requirements are estimated. The advantage of this sort of algorithm is the simplicity in terms of both implementation as well as overhead, since there is no need to constantly monitor the workstations for performance statistics.

Figure 3.1: Static Load Balancing
However, static algorithms only work well when there is not much variation in the load on the workstations. A few static load balancing techniques are:

- **Round robin algorithm** - the tasks are passed to processes in a sequential order; when the last process has received a task the schedule continues with the first process (a new round).

- **Randomized algorithm**: the allocation of tasks to processes is random

- Simulated annealing or genetic algorithms: mixture allocation procedure including optimization techniques

Even when a good mathematical solution exists, static load balancing still have several flaws:

- it is very difficult to estimate a-priori [in an accurate way] the execution time of various parts of a program

- sometimes there are communication delays that vary in an uncontrollable way

- for some problems the number of steps to reach a solution is not known in advance

**(b) Dynamic Load Balancing Algorithm**

Dynamic load balancing algorithms make changes to the distribution of work among workstations at run-time; they use current or recent load information when making distribution decisions.
Multicomputers with dynamic load balancing allocate reallocate resources at runtime based on no a priori task information, which may determine when and whose tasks can be migrated [57, 58]. As a result, dynamic load balancing algorithms can provide a significant improvement in performance over static algorithms. However, this comes at the additional cost of collecting and maintaining load information, so it is important to keep these overheads within reasonable limits.

**3.3.2: Load Balancing Strategies**

There are three major parameters which usually define the strategy a specific load balancing algorithm will employ. These three parameters answer three important questions:

- Who makes the load balancing decision
- What information is used to make the load balancing decision,
- Where the load balancing decision is made.
**Sender-Initiated vs. Receiver-Initiated Strategies**

The question of who makes the load balancing decision is answered based on whether a sender-initiated or receiver-initiated policy is employed [59]. In sender-initiated policies, congested nodes attempt to move work to lightly-loaded nodes.

![Performance of Sender-initiated Vs Receiver-initiated Strategies](image)

**Figure 3.3: Performance of Sender-initiated Vs Receiver-initiated Strategies**

NLS – No Load Sharing, SI – Sender Initiated, RI – Receiver Initiated.

In receiver-initiated policies, lightly-loaded nodes look for heavily-loaded nodes from which work may be received.

Figure 3.3 shows the relative performance of a sender-initiated and receiver-initiated load balancing algorithm. As can be seen, both the sender-initiated and receiver-initiated policies perform substantially better than a system which has no load sharing.

The sender-initiated policy performs better than the receiver-initiated policy at low to moderate system loads. Reasons are that at these loads, the probability of finding a
lightly-loaded node is higher than that of finding a heavily-loaded node. Similarly, at high system loads, the receiver initiated policy performs better since it is much easier to find a heavily-loaded node. As a result, adaptive policies have been proposed which behave like sender-initiated policies at low to moderate system loads, while at high system loads they behave like receiver-initiated policies.

**Global vs. Local Strategies**

Global or local policies answer the question of what information will be used to make a load balancing decision in global policies, the load balancer uses the performance profiles of all available workstations. In local policies workstations are partitioned into different groups. The benefit in a local scheme is that performance profile information is only exchanged within the group. The choice of a global or local policy depends on the behaviour an application will exhibit. For global schemes, balanced load convergence is faster compared to a local scheme since all workstations are considered at the same time. However, this requires additional communication and synchronisation between the various workstations; the local schemes minimise this extra overhead. But the reduced synchronisation between workstations is also a downfall of the local schemes if the various groups exhibit major differences in performance. If one group has processors with poor performance (high load), and another group has very fast processors (little or no load), the latter will finish quite early while the former group is overloaded.

**Centralized vs. De-centralized Strategies**

A load balancer is categorized as either centralized or distributed, both of which define where load balancing decisions are made [60-64]. In a centralized scheme, the load
balancer is located on one master workstation node and all decisions are made there.

Basic features of centralized approach are:

- A master node holds the collection of tasks to be performed
- Tasks are sent to the execution node
- When a executing process completes one task, it requests another task from the master node

![Manager Node](image)

**Figure 3.4: Centralized Strategies**

In a de-centralized scheme, the load balancer is replicated on all workstations. There are different algorithms used in de-centralized scheme for job selection. These algorithms are Round robin algorithm, Random polling algorithm etc. The scheme used in these algorithms is pictorially depicted in figure 3.5.

Once again, there are tradeoffs associated with choosing one location scheme over the other. For centralized schemes, the reliance on one central point of balancing control could limit future scalability.
Figure 3.5: Decentralized Strategies

Additionally, the central scheme also requires an “all-to-one” exchange of profile information from workstations to the balancer as well as a “one-to-all” exchange of distribution instructions from the balancer to the workstations. The distributed scheme helps solve the scalability problems, but at the expense of an “all-to-all” broadcast of profile information between workstations. However, the distributed scheme avoids the “one-to-all” distribution exchange since the distribution decisions are made on each workstation.

3.3.3: Load Balancing Policies

**Transfer policy**: First of all the state of the different machines is determined by calculating it’s workload. A transfer policy determines whether a machine is in a suitable state to participate in a task transfer, either as a sender or a receiver. For example, a heavily loaded machine could try to start process migration when its load index exceeds a certain threshold.
**Selection policy:** This policy determines which task should be transferred. Once the transfer policy decides that a machine is in a heavily-loaded state, the selection policy selects a task for transferring. Selection policies can be categorized into two policies: preemptive and non-preemptive. A preemptive policy selects a partially executed task. As such, a preemptive policy should also transfer the task state which can be very large or complex. Thus, transferring operation is expensive. A non-preemptive policy selects only tasks that have not begun execution and, hence, it does not require transferring the state of task.

**Location policy:** The objective of this policy is to find a suitable transfer partner for a machine, once the transfer policy has decided that the machine is a heavily-loaded state or lightly-loaded one. Common location policies include: random selection, dynamic selection, and state polling.

**Information policy:** This policy determines when the information about the state of other machines should be collected, from where it has to be collected, and what information is to be collected.

**3.4: LITERATURE REVIEW**

Load balancing is an important activity in all types of distributed systems. At the service level two essential functions in a grid infrastructure are workload and resource management. The main issues in computational grids are heterogeneity, scalability and adaptability. In [65] a layered algorithm which achieves dynamic load balancing has been discussed. The main features of the model in this paper are layered architecture, scalability and heterogeneity and independance from the physical architecture of the grid.
Two job migration algorithms for a grid architecture MELISA and LBA are discussed in [66]. Load balancing is carried out in different ways in these two algorithms and they minimize the response time on large and small scale heterogeneous Grid environments, respectively. MELISA is a modified version of ELISA [67], and is applicable to large scale systems. In this algorithm resource heterogeneity, job migration cost and network heterogeneity are considered. The LBA algorithm is applicable to small scale systems and load balancing is performed in it by estimating the expected finish time of a job on buddy processors. In [68] the problem of scheduling and load balancing in a grid architecture with dispersed computational resources in different administrative domains is considered. Min-load and Min-cost policies are considered in this paper for scheduling jobs in multi-cluster environment. A load balancing algorithm for fair scheduling in a grid environment has been discussed in [69]. The mean waiting time has been used in this work to address the issue of fairness. The execution time is reduced here as compared to other algorithms such as Earliest Deadline First, Simple Fair Task Order, Adjusted Fair Task Order and Max Min Fair Scheduling. Along with this the expected price for execution of all the jobs in the grid system is also minimized.

A Novel Load Balancing Algorithm based on system heterogeneity and migration of I/O intensive tasks from heavily loaded to lightly loaded nodes in clusters is discussed in [70]. In this algorithm both the average slowdown of all parallel jobs running on a cluster and average response time are reduced. Load Balancing in Linux Cluster is discussed in [71]. One node from the cluster is considered as Master Server and another is considered as Load Balancer. The IP and CPU information of each machine is maintained at the Master
Server which is periodically sent from each node to the Master Server. This algorithm can solve the Readers and writers problem using sockets.

In [72] a practical and cohesive framework for load balancing that improves upon existing strategies is discussed. These techniques can be implemented on various architectures such as shared memory systems, networks of workstations and massively parallel machines. Here a task selection mechanism which can preserve or improve communication locality is implemented. Proper initial placement of processes as well as migration of processes to processors with fewer resident processes is very critical in loosely coupled homogeneous processors. In [73] the behaviour of a large number of UNIX processes is analyzed and was then used to drive simulation studies of particular dynamic assignment heuristics. A non-linear time delay model of the load balancing algorithm has been studied in [74]. The constraints imposed by the effect of delays in the exchange of information among nodes in real environments is considered. These effects have been studied over two different test beds: Local Area Network and Planet-Lab. The results obtained from the two test beds has been found to be consistent. A dynamic load balancing algorithm based on the operation of a single-queue multiserver queueing system is developed in [75]. The policy used here is known as LBC which has a central job dispatcher. This policy provides minimum average job response time for small mean job transfer delay and it’s performance is not affected by heterogeneous load and hence it can adapt to changing load naturally. A method for improving the performance in queuing systems is also discussed in [76]. In this paper the goal is to maximize the fraction of jobs that meet their deadlines.
In [77] a general discussion of dynamic load balancing on Web-Server systems is provided. Qualitative parameters have been compared for load balancing in Distributed Web Server Systems in [78]. An autonomous load balancing system using active object model has been implemented in [79] for distributed servers. It is necessary for the web servers to service the client request quickly in order to minimize the client response time. An algorithm called SWIFT has been implemented in [80] which focuses on both server and network characteristics to achieve this goal.

Simulation system for load balancing algorithms for workstation based environment has been developed in [81]. A discrete event simulation system called DiMSIM is developed on a local area network of DEC stations. A new algorithm called balanced join shortest queue is compared in this work with three existing algorithms and the results obtained validate the suggestions made by [82] that complex usage of state information are not of much benefit. The concept of virtual routing has been used in [83] for minimizing the mean response time of a multi class job. A static decentralized algorithm has been developed in this paper. Another load balancing algorithm for multi-class, multi-server processor-sharing system with a poisson distribution of incoming processes is investigated in [84].

A diffusive load balancing algorithm has been simulated in three different client server environments in [85] and has been compared with the existing algorithms. Three policies namely rate adjustment policy, queue adjustment policy and Queue and Rate adjustment policy can be used to classify dynamic, decentralized algorithms. Based upon these three policies algorithms have been developed and compared in [86] and suitability of these under various situations has been discussed.
The complexity for finding out the workload information for \( v \) number of nodes is \( O(v^2) \), where \( v \) is the number of nodes. A special adjacency matrix has been designed in [87] which reduces the complexity to \( O(v\sqrt{v}) \) by eliminating the redundant information collected in the previous algorithms.

Knowledge based techniques have also been used by many researchers to design adaptive and dynamic solutions for load balancing. One such work is presented in [88] where heuristic knowledge and real time decision making is used to manage dynamic systems in a decentralized manner.