DISTRIBUTED SYSTEM MANAGEMENT

MCA Sem IV
Paper Code-CS4T16

-- Dr. Bhawna Sinha
Department of MCA,
PWC

Introduction

- Distributed systems have multiple resources and hence, there is a need to provide systems transparency
- One of the functions of a distributed operating system is to assign processes to the nodes (resources) of the distributed system such that the resource usage, response time, network congestion, and scheduling overhead are optimized.
- System management can be categorized into resource management, process management, and fault tolerance.

Scheduling Techniques in Distributed System

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

Scheduling Techniques

Task Assignment Approach, in which 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.

Load-balancing approach, in which all the processes submitted by the users are distributed among the nodes of the system so as to equalize the workload among the nodes.

Load-sharing approach, which simply attempts to conserve the ability of the system to perform work by assuring that no node is idle while processes wait for being processed.

Desirable Features of Global Scheduling Algorithm

- No A Priori knowledge about the Processes
- Ability to make dynamic scheduling decisions
- Flexible
- Stable
- Balanced System Performance and Scheduling Overhead
- Unaffected by system failures
- Scalable

No A Priori knowledge about the Processes

In computing, scheduling is the method by which threads, processes or data flows are given access to system resources (e.g. processor time, communications bandwidth). This is usually done to load balance and share system resources effectively or achieve a target quality of service.

Scheduling algorithms that operate based on the information about the characteristics and resource requirements of the processes pose an extra burden on the users who must provide this information while submitting their processes for execution.

A good process scheduling algorithm should operate with absolutely no a priori knowledge about the processes to be executed. Since it places extra burden on the user to specify this information before execution

Ability to make dynamic scheduling decisions

- Process assignment decisions should be dynamic, i.e., be based on the current load of the system and not on some static policy. It is recommended that the scheduling algorithm possess the flexibility to migrate a process more than once because the initial decision of placing a process on a particular node may have to be changed after some time to adapt to the new system load.
- A good process scheduling algorithm should be able to take care of the dynamically changing load (or status) of the various nodes of the system.

Flexible

- The algorithm should be flexible enough to migrate the process multiple times in case there is a change in the system load.

- The algorithm should be able to make quick scheduling decisions about assigning processes to processors.

Stability

- The algorithm must be stable such that processors do useful work, reduce thrashing overhead and minimize the time spent in unnecessary migration of the process.

- **Example:** it may happen that node *n*1 and *n*2 both observe that node *n*3 is idle and then both offload a portion of their work to node *n*3 without being aware of the offloading decision made by the other. Now if node *n*3 becomes overloaded due to the processes received fro both nodes *n*1 and *n*2 , then it may again start transferring its processes to other nodes. This entire cycle may be repeated again and again, resulting in an unstable state. This is certainly not desirable for a good scheduling algorithm.

Balanced system performance and scheduling overhead

Algorithms that provide near-optimal system performance with a minimum of global state information (such as CPU load) gathering overhead are desirable. This is because the overhead increases as the amount of global state information collected increases. This is because the usefulness of that information is decreased due to both the aging of the information being gathered and the low scheduling frequency as a result of the cost of gathering and processing the extra information.

Unaffected by system failures

- A good scheduling algorithm should not be disabled by the crash of one or more nodes of the system. Also, if the nodes are partitioned into two or more groups due to link failures, the algorithm should be capable of functioning properly for the nodes within a group. Algorithms that have decentralized decision making capability and consider only available nodes in their decision making have better fault tolerance capability.

Scalability

A scheduling algorithm should scale well as the number of nodes increases. An algorithm that makes scheduling decisions by first inquiring the workload from all the nodes and then selecting the most lightly loaded node has poor scalability. This will work fine only when there are few nodes in the system. This is because the inquirer receives a flood of replies almost simultaneously, and the time required to process the reply messages for making a node selection is too long as the number of nodes (N) increase. Also the network traffic quickly consumes network bandwidth. A simple approach is to probe only m of N nodes for selecting a node.

TASK ASSIGNMENT APPROACH

Each process is divided into multiple tasks. These tasks are scheduled to suitable processor to improve performance. This is not a widely used approach because:

- It requires characteristics of all the processes to be known in advance.
- This approach does not take into consideration the dynamically changing state of the system.

Goal of the Task Assignment Approach

In this approach, a process is considered to be composed of multiple tasks and the goal is to find an optimal assignment policy for the tasks of an individual process.

- Minimization of IPC costs
- Quick turnaround time for the complete process
- A high degree of parallelism
- Efficient utilization of system resources in general

These goals often conflict. E.g., while minimizing IPC costs tends to assign all tasks of a process to a single node, efficient utilization of system resources tries to distribute the tasks evenly among the nodes.

Also note that in case of m tasks and q nodes, there are mq possible assignments of tasks to nodes . In practice, however, the actual number of possible assignments of tasks to nodes may be less than mq due to the restriction that certain tasks cannot be assigned to certain nodes due to their specific requirements (e.g. need a certain amount of memory or a certain data file).

Assumptions For Task Assignment Approach

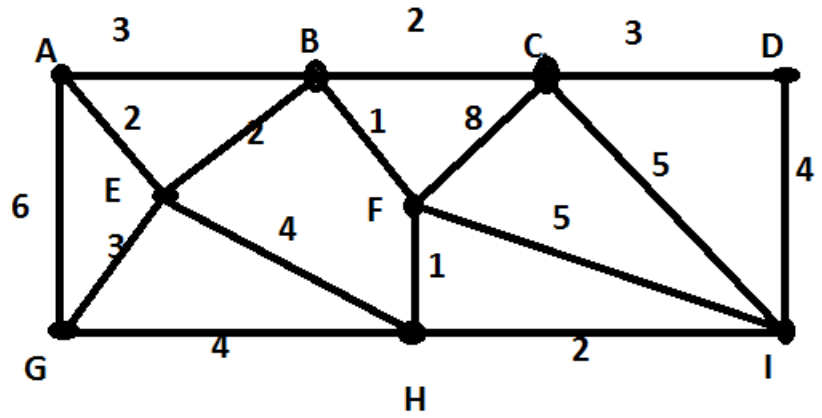
- A process has already been split into pieces called tasks.
- The amount of computation required by each task and the speed of each processor are known.
- The cost of processing each task on every node of the system is known.
- The Interprocess Communication (IPC) costs between every pair of tasks is known.
- Other constraints, such as resource requirements of the tasks and the available resources at each node, precedence relationships among the tasks, and so on, are also known.

Task Assignment Approach Algorithms

- **Graph Theoretic Deterministic Algorithm.**
- **Centralized Heuristic Algorithm.**
- **Hierarchical Algorithm.**

Graph Theoretic Deterministic Algorithm

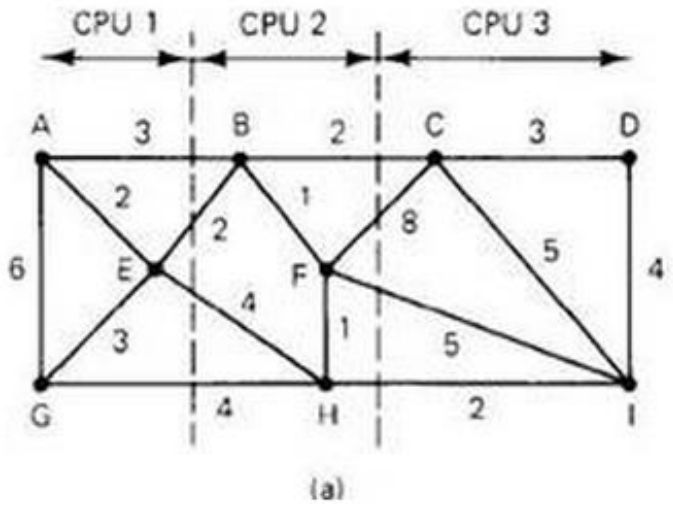
- This algorithm requires a system consisting of processes with known CPU and memory requirements, and a known matrix giving the average amount of traffic between each pair of processes. If the number of CPUs, *k,* is smaller than the number of processes, several processes will have to be assigned to each CPU. The idea is to perform this assignment such as to minimize network traffic.



- Optimal assignments are found by first creating a static assignment graph. In this graph, the weights of the edges joining pairs of task nodes represent inter-task communication costs. The weight on the edge joining a task node to node n1 represents the execution cost of that task on node n2 and vice-versa. Then we determine a minimum cutset in this graph. A cutset is defined to be a set of edges such that when these edges are removed, the nodes of the graph are partitioned into two disjoint subsets such that nodes in one subset are reachable from n1 and the nodes in the other are reachable from n2. Each task node is reachable from either n1 or n2. The weight of a cutset is the sum of the weights of the edges in the cutset. This sums up the execution and communication costs for that assignment. An optimal assignment is found by finding a minimum cutset.

Graph Theoretic Deterministic Algorithm Cont.

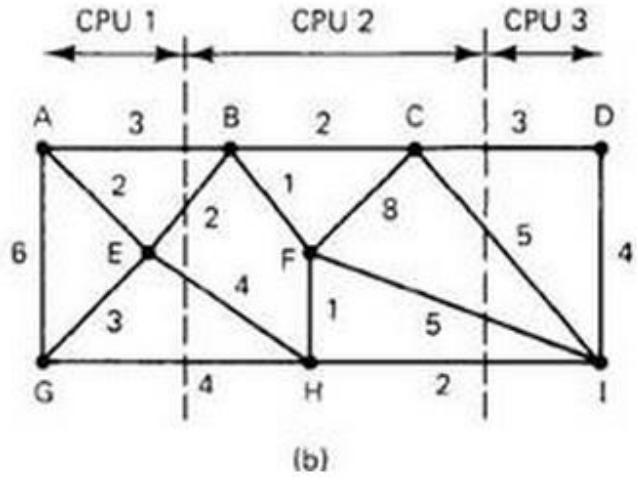
- Example 1:



- Network Traffic = 30

Graph Theoretic Deterministic Algorithm Cont.

- Example 2:

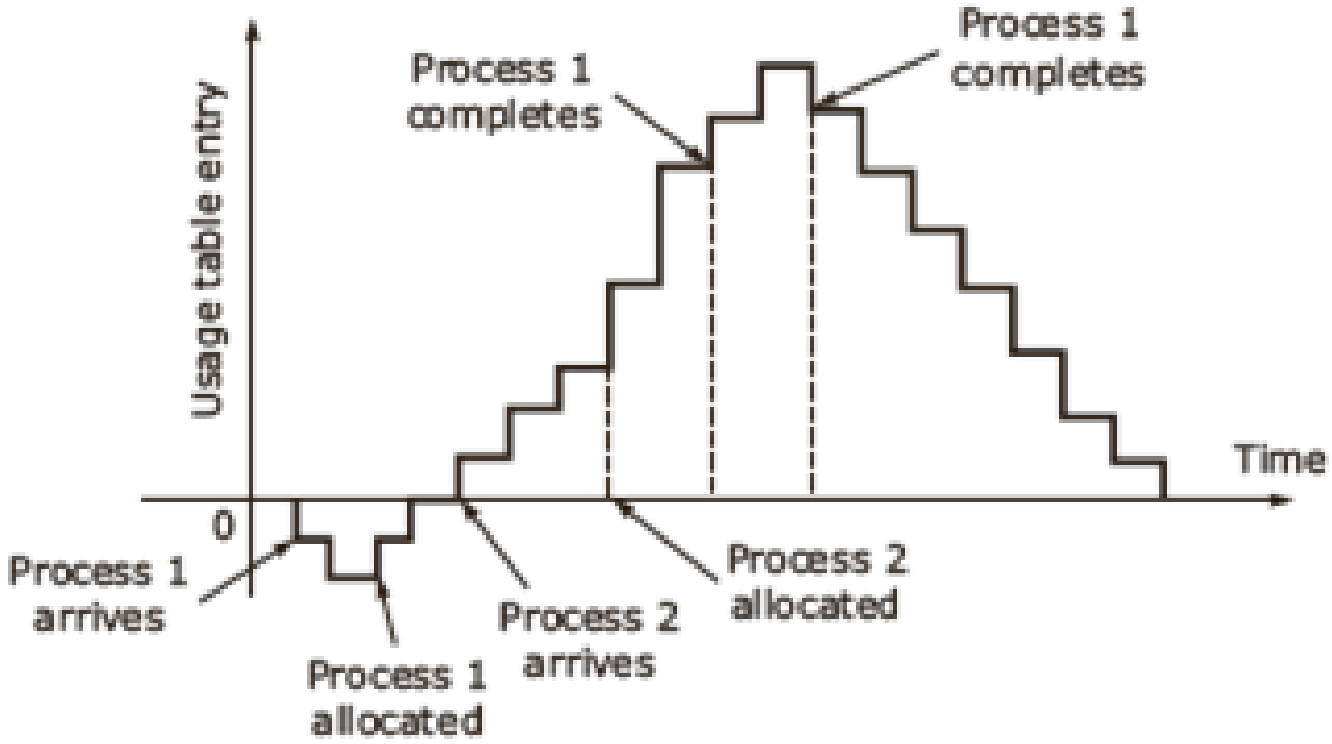


- Network Traffic = 28

Centralized Heuristic Algorithm

- Also called Top down algorithm
- Doesn’t require advance information
- Coordinator maintains the usage table with one entry for every user (processor) and this is initially zero. When significant events happen, messages sent to the coordinator and the table is updated.
- Usage table entries can either be zero, positive, or negative. Zero value indicates a neutral state, a positive value implies that the machine is user of system resources, and a negative value means that the machine needs resources.
- The heuristic used for processor allocation is that when the CPU becomes free, pending requests whose owners have the lowest score win. As a result, a user who has a request pending for a long time will always be allocated a processor first.

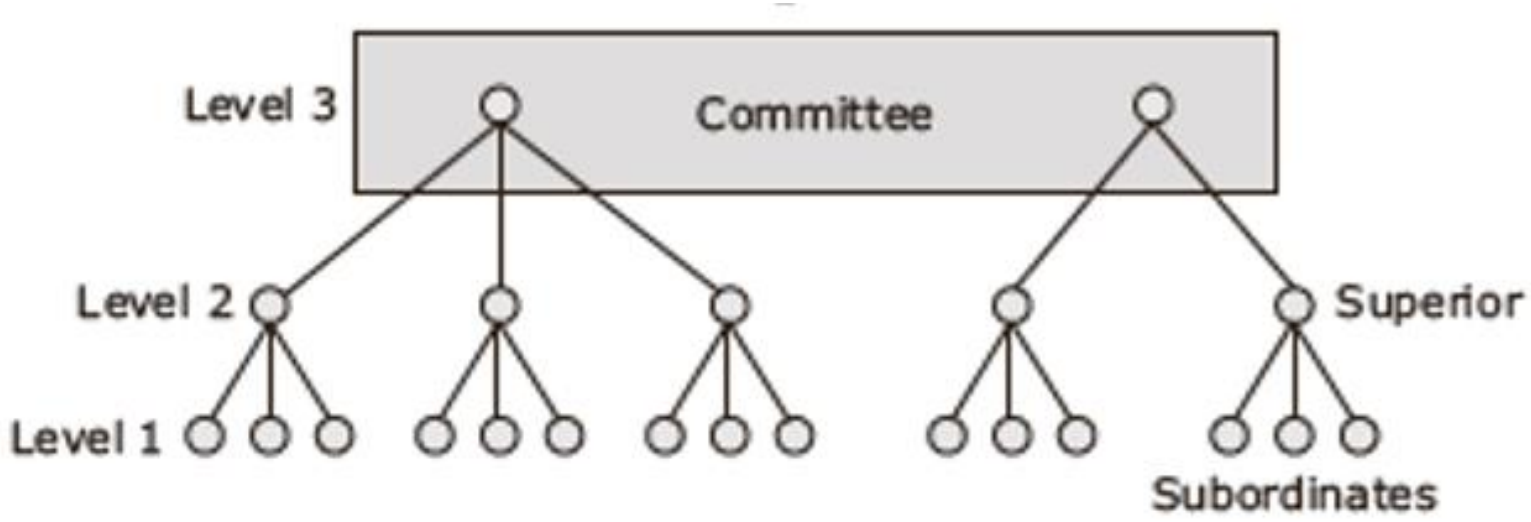
Centralized Heuristic Algorithm Cont.



Hierarchical Algorithm

- Centralized algorithms, such as up-down, do not scale well to large systems. The central node soon becomes a bottleneck
- This problem can be attacked by using a hierarchical algorithm instead of a centralized one.
- As the number of processors is increased, the number of level is increased.
- For each group of *k* workers, one manager machine is assigned the task of keeping track of who is busy and who is idle. Each processor maintains communication with one superior and a few subordinates.
- The information stream is managable, but the system could fail if any middle level machine fails.
- The allocation algorithm works between two levels in the group. The situation is complex because at any instant of time, multiple requests are in various stages of allocation, leading to outdated available worker estimates.

Hierarchical Algorithm



3.3 Load Balancing Approach

- Processing speed of a system is always highly intended.
- Distributed computing system provides high performance environment that are able to provide huge processing power.
- In distributed computing thousand of processors can be connected either by wide area network or across a large number of systems which consists of cheap and easily available autonomous systems like workstations or PCs.
- The distribution of loads to the processing elements is simply called the load balancing.
- The goal of the load balancing algorithms is to maintain the load to each processing element such that all the processing elements become neither overloaded nor idle that means each processing element ideally has equal load at any moment of time during execution to obtain the maximum performance (minimum execution time) of the system.

Load Balancing

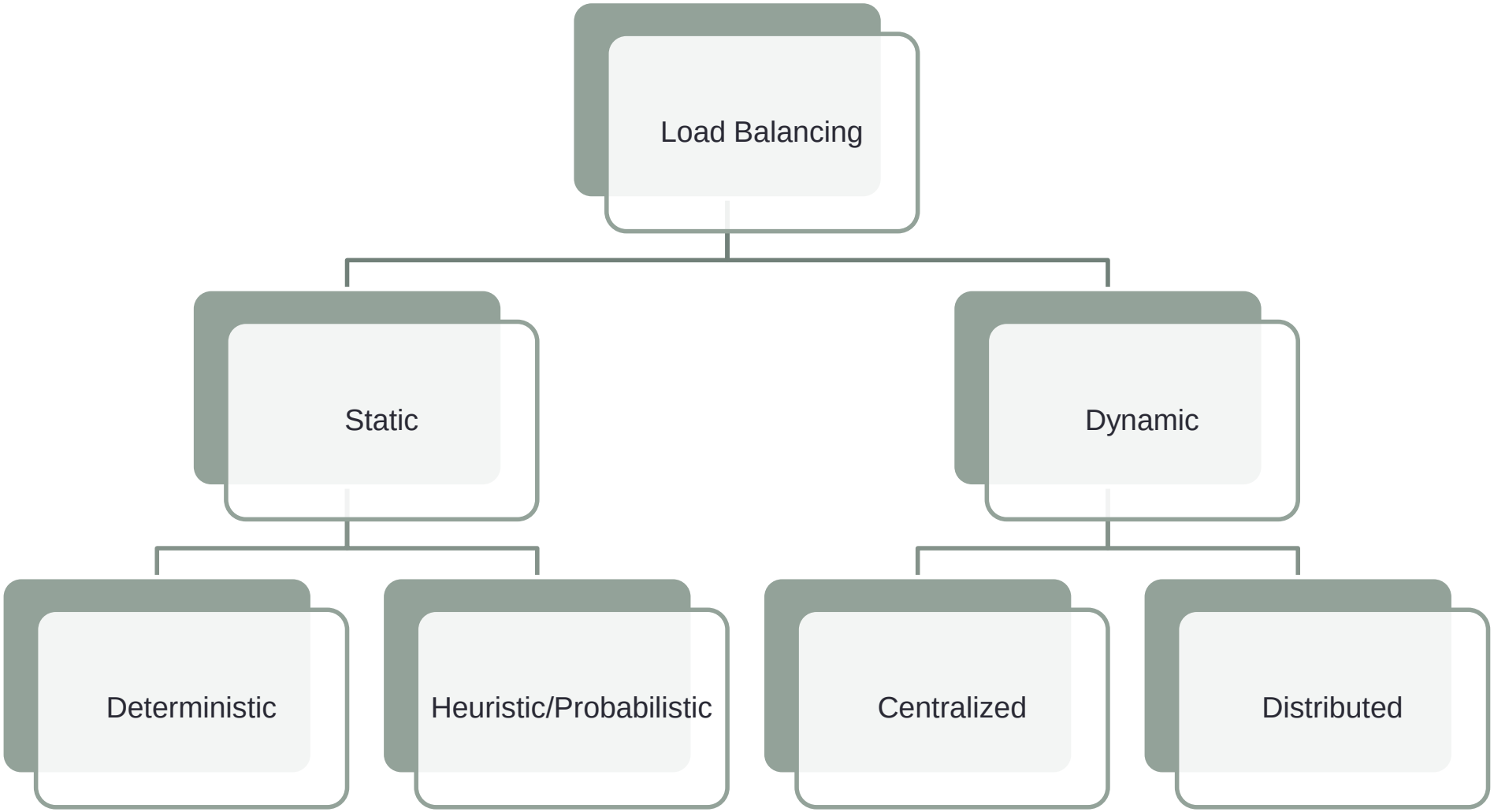
- Load balancing is the way of distributing load units (jobs or tasks) across a set of processors which are connected to a network which may be distributed across the globe.

- The excess load or remaining unexecuted load from a processor is migrated to other processors which have load below the threshold load.

- Threshold load is such an amount of load to a processor that any load may come further to that processor.

- By load balancing strategy it is possible to make every processor equally busy and to finish the works approximately at the same time.

Taxonomy Of Load Balancing



Static Load Balancing

- In static algorithm the processes are assigned to the processors at the compile time according to the performance of the nodes.
- Once the processes are assigned, no change or reassignment is possible at the run time.
- Number of jobs in each node is fixed in static load balancing algorithm. Static algorithms do not collect any information about the nodes .

Sub Classes of SLB

- The static load balancing algorithms can be divided into two sub classes:
- Optimal static load balancing (Deterministic)
- Sub optimal static load balancing (Probabilistic)

- Optimal Static Load Balancing Algorithm
- If all the information and resources related to a system are known optimal static load balancing can be done such as the list of processes, computing requirements, file requirements and communication requirements

Sub Optimal Static Load Balancing Algorithm

- Sub-optimal load balancing algorithm will be mandatory for some applications when optimal solution is not found.
- In case the load is unpredictable or variable from minute to minute or hour to hour.

Dynamic Load Balancing

- In dynamic load balancing algorithm assignment of jobs is done at the runtime.
- In DLB jobs are reassigned at the runtime depending upon the situation that is the load will be transferred from heavily loaded nodes to the lightly loaded nodes.
- In dynamic load balancing no decision is taken until the process gets execution.
- This strategy collects the information about the system state and about the job information.
- As more information is collected by an algorithm in a short time, potentially the algorithm can make better decision.

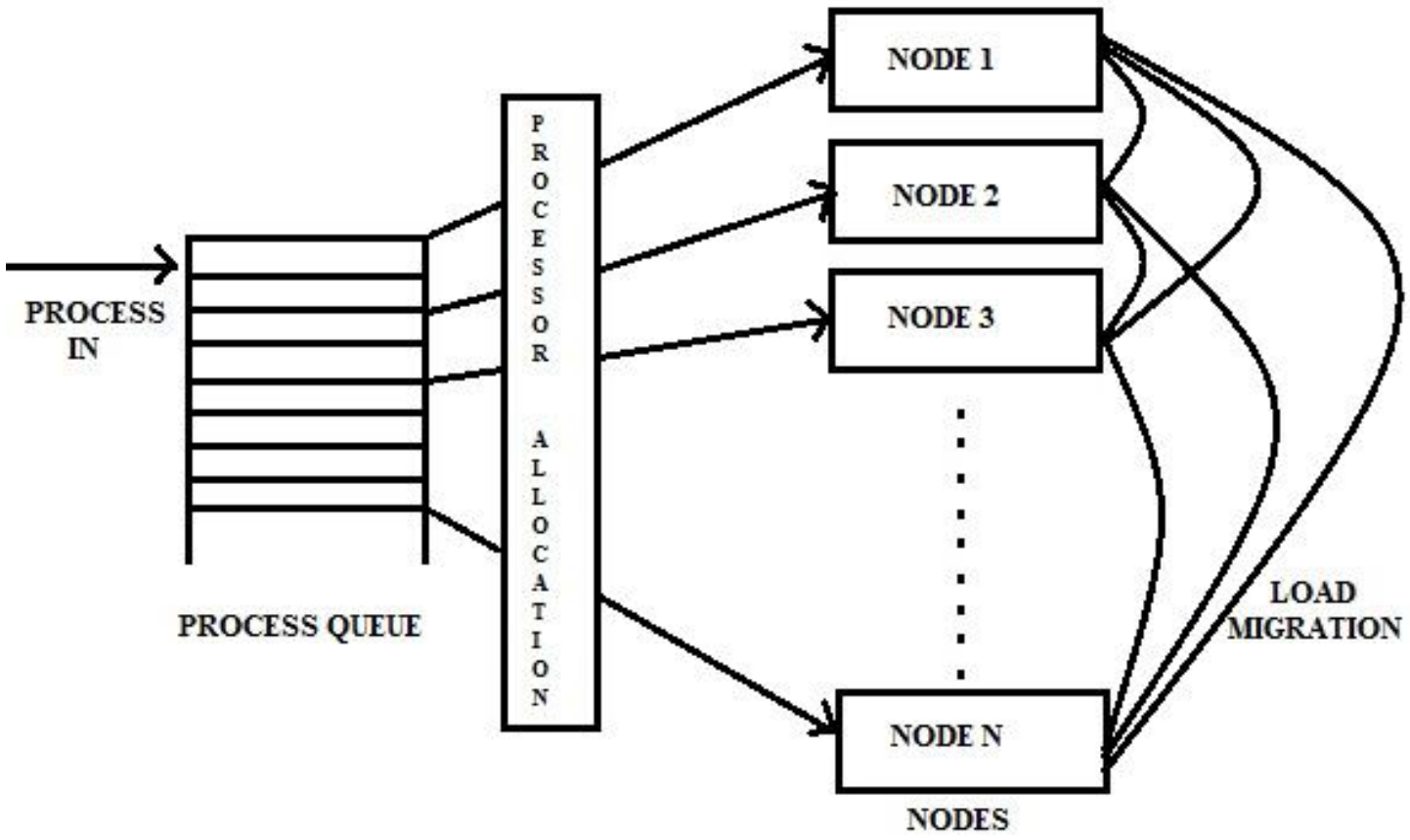


Figure: Job Migration in Dynamic Load Balancing Strategy

Centralized Vs Distributed

- A **centralized** dynamic scheduling algorithm means that the scheduling decision is carried out at one single node called the centralized node. This approach is efficient since all information is available at a single node.
- Drawback for **centralized** approach is it leads to a bottleneck as number of requests increase .
- In a **distributed** algorithm the task of processor assignment is physically distributed among various nodes .
- In **distributed algorithm** scheduling decisions are made on individual nodes.

Cooperative Vs non-cooperative

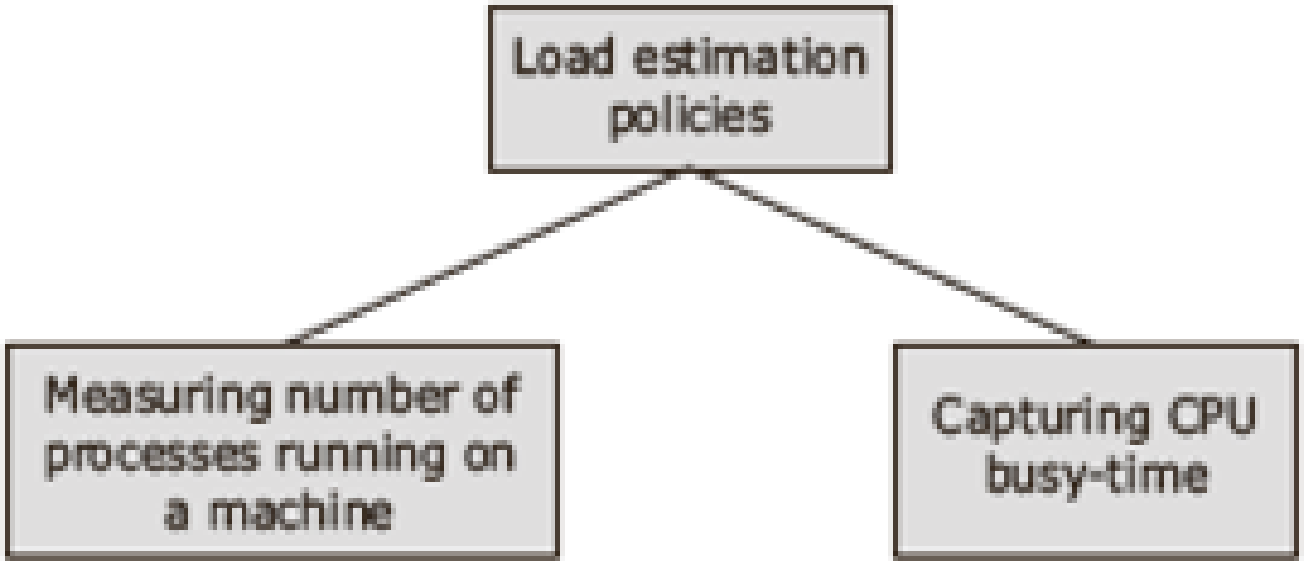
- **Cooperative Algorithm** distributed entities cooperate with each other to make scheduling decisions
- **Non-Cooperative algorithm** the individual entities make independent scheduling decisions and hence they involve minor overheads
- Cooperative algorithms are more complex than non-cooperative ones
- Non-cooperative algorithms may not be stable

Issues in designing in load balancing algorithms

- Deciding policies for:
  - **Load estimation:** determines how to estimate the workload of a node in a distributed system.
  - **Process transfer:** decides whether the process can be executed locally or there is a need for remote execution.
  - **Static information exchange:** determines how the system load information can be exchanged among the nodes.
  - **Location Policy:** determines the selection of a destination node during process migration
  - **Priority assignment:** determines the priority of execution of a set of local and remote processes on a particular node

Policies for Load estimation

- Parameters:
  - Time dependent
  - Node dependent



Measuring Number of processes running on a machine

- One way to estimate the load is to calculate the number of processes running on the machine
- The process count is not the correct answer for load calculation because the machine can have many processes running, such as mail, news, windows managers, etc both in foreground and background

Capturing CPU busy time

- The other technique is to capture the CPU busy-time.
- A machine with 75% CPU utilization is more heavily loaded than a machine with 40% CPU utilization.

- CPU utilization can be measured be measured by allowing a timer to interrupt the machine to periodically observe the CPU state and find the fraction of idle time.

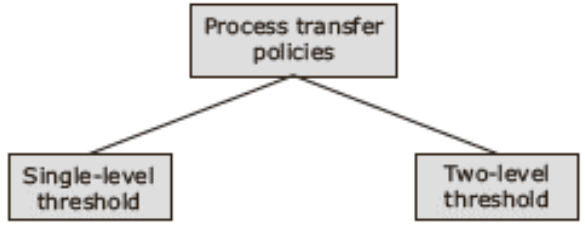
Policies for Process transfer

- Load balancing strategy involves transferring some processes from heavily loaded nodes to lightly nodes
- So there is a need to decide a policy which indicates whether a node is heavily or lightly loaded, called threshold policy.
- This threshold is a limiting value which decides whether a new process, ready for execution or transferred to a lightly loaded node
- Threshold policy
  - Static
  - Dynamic

Threshold Policies

- Threshold policy
  - **Static:** each node has a predefined threshold value depending on its processing capability, which does not vary with load. The advantage of this method is that there is no need for any exchange of state information to decide the threshold value
  - **Dynamic:** the threshold value is calculated as an average workload of all nodes. This policy gives a realistic value but involves the overhead of state information exchange among nodes to determine the threshold value.

Threshold Policy Cont.

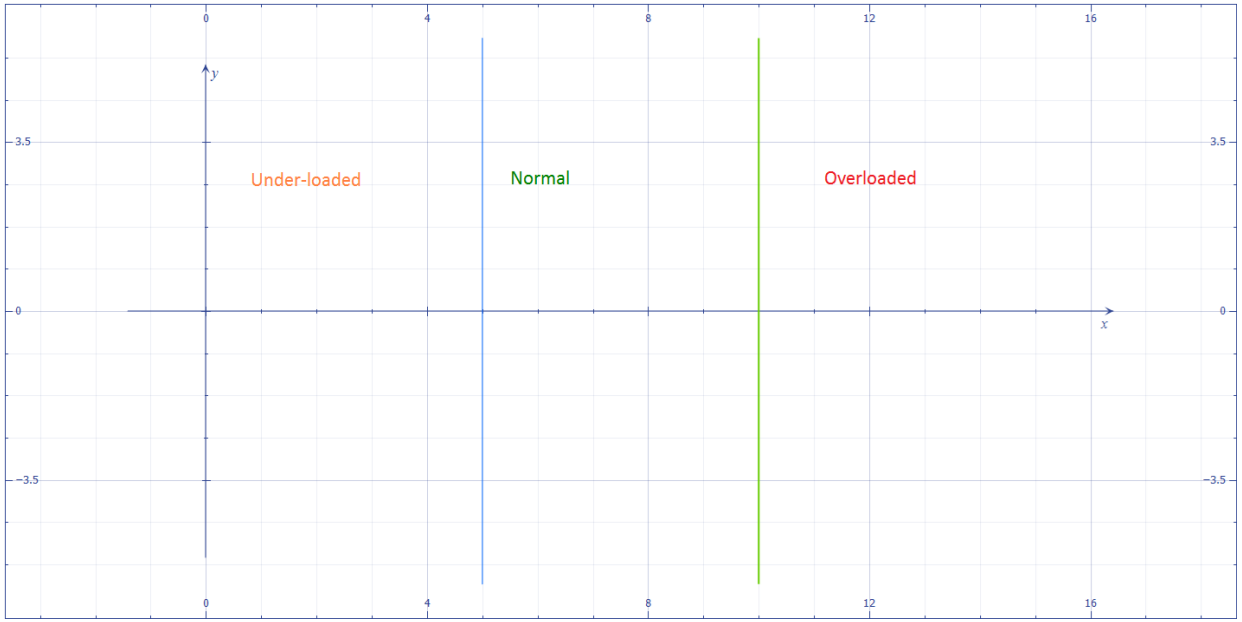


- **Single-level threshold:** a node accepts a new process as long as its load is below the threshold, else it rejects the process as well as the requests for remote execution.

- The use of this policy may lead to useless process transfer, leading to instability in decisions. This may happen in a situation where a node accepts a new or a remote process when the load is below threshold , but its load may become larger than threshold just when the remote process arrives for execution.

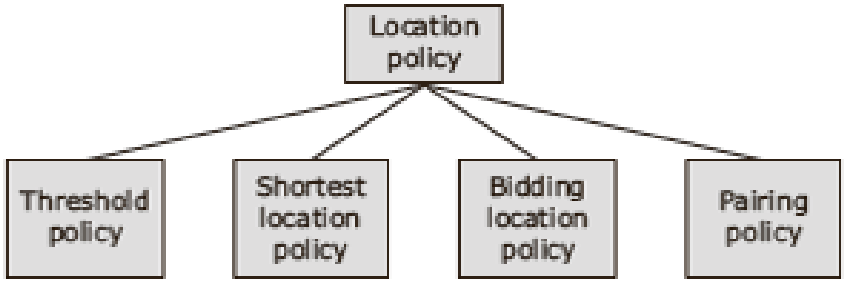
Threshold Policy Cont.

- **Two-level threshold policy:** this policy is preferred to avoid instability it has two threshold levels: high and low marks. The load states of a node can be divided into three regions: overloaded, normal and under-loaded



Location policies

- Once the threshold transfer policy decides to transfer a process from a node, the next step is to use a location policy to select the destination node where the process can be executed.



Threshold policy

- In the threshold policy, the destination node is selected at random and a check is made to verify whether the remote process transfer would load that node, If not, the process transfer is carried out; else another node is selected at random and probed. This process continues till a suitable destination node is found or the number of nodes probed exceeds a probe limit(defined by the system)

Shortest Location policy

- Nodes are chosen at random and each of these nodes is polled to check for load.
- The node with the lowest load value is selected as the destination node.
- Once selected the destination node has to execute the process irrespective of its state at the time the process arrives.
- In case non of the polled nodes can accept the process it will be executed at the source node itself

Bidding Location Policy

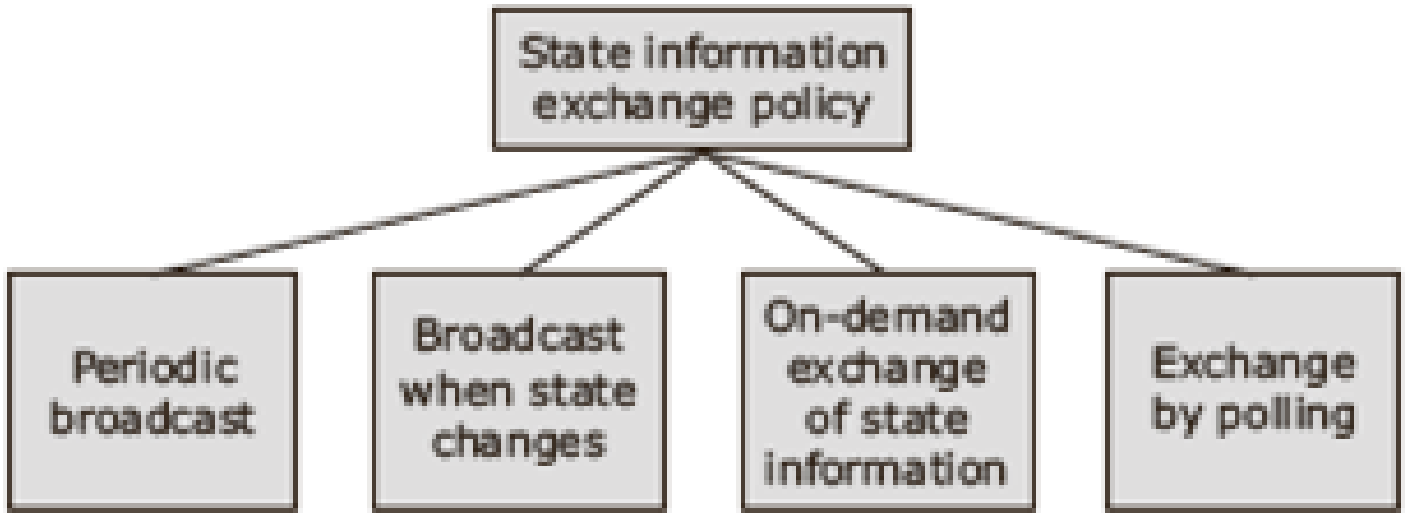
- This policy transforms the system into a market scenario with buyers and sellers of services.
- Each node is assigned two roles, namely the manager and the contractor.
- The manager is an under loaded node having a process which needs a location and the contractor is a node which can accept remote processes.
- The manager broadcasts a request for a bid message to all nodes and the contractors send their bids to the manager node
- The bid contains information about processing power, and memory size
- The manager chooses the best bid which is the cheapest, and fastest then the process transferred to the winning contractor node.
- If the contractor won too many bids at a time from many managers this may become overload, so when the best bid is selected a message sent to the owner of the bid which can weather accept or reject the process

Pairing Policy

- The policies discussed earlier focus on load balancing across the systems, while the pairing policy focuses on load balancing between a pair of nodes
- Two nodes which have a large difference of load balancing between a pair of nodes are paired together temporarily
- The load balancing is carried out between the nodes belonging to the same pair by migrating processes from the heavily loaded node to the lightly loaded node
- Several pairs of nodes can exist in the system simultaneously

State information exchange

- Dynamic policies require frequent exchange of state information among the nodes of a system
- Decision based on state information



Periodic Broadcast

- In this policy, each node broadcasts its state information at time interval t .
- This method generates heavy traffic and there may be unnecessary messages transmitted in case where the node state has not changed in time t .
- Scalability is also an issue, since the number of messages generated for state information exchanges will be too large for a network with many nodes.

Broadcast when state changes and on-demand exchange of state information

- **Broadcast when state changes:** In this method, a node broadcasts its state information only when a node's state changes (when a process arrives at a node or a process departs from a node).

- **On-demand exchange of state information:** The method of on-demand exchange of state information is based on the fact that a node needs to know the state of other nodes, only when it is either **under-loaded or overloaded.**
  - A node will broadcast $Stateinformationrequest$ message then the nodes send their current state to the requesting node

Exchange by polling

- Broadcasting occurs only when a node needs cooperation from another node for load balancing
- It search for a suitable partner by polling all nodes one by one and exchanging state information
- The polling process is stopped when a suitable pair is found or a predefined poll limit is reached

3.4 Load Sharing Approach

- Load balancing approaches attempt to equalize the workload on all the nodes of a system by gathering state information

- Load sharing algorithms do not attempt to balance the average workload on all nodes, they only ensure that no node is idle or heavily loaded

- Policies for load sharing approach are the same as load balancing poli ion policy, process transfer policy rmation exchange, they differ in l

Location sharing policies

Location policy for load sharing approach

- **Sender Initiated algorithm** uses sender of the process to decide where to send the process
  - The heavily loaded node search for lightly loaded nodes where the process can be transferred
  - When a load on a node increases beyond the threshold , it probes nodes to find a lightly loaded node .
  - A node can receive a process only if its transfer will not increase its load beyond the threshold .
  - If a suitable node is not found the process will be executed on the same node.

Location policy for load sharing approach cont.

- **Receiver initiated location policy**
  - In this policy lightly loaded nodes search for heavily loaded nodes from which processes can be accepted for execution .
  - When the load on a node falls below a threshold value, it broadcasts a probe message to all nodes or probes nodes one by one to search for a heavily loaded node
  - A node can transfer one of its processes if such a transfer does not reduce its load below normal threshold