● Load balancing approach

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.

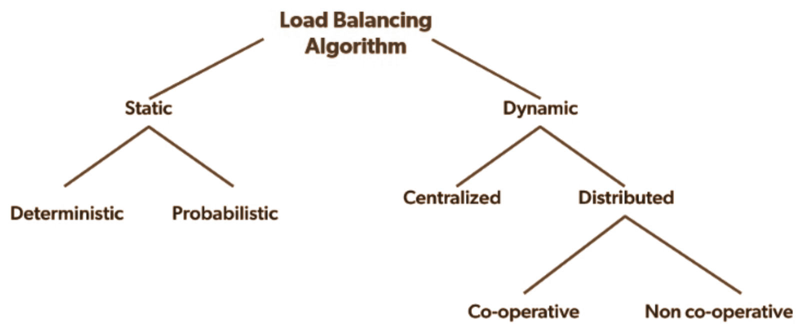
Threshold load is such an amount of load to a processor that any load may come further to that processor.

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

<u>Purpose of Load Balancing in Distributed Systems:</u>

- Security: A load balancer provides safety to your site.
- Protect applications from emerging threats: The Web Application Firewall (WAF) in the load balancer shields your site.
- Authenticate User Access: The load balancer can demand a username and secret key prior to conceding admittance to your site to safeguard against unapproved access.
- Protect against DDoS attacks: The load balancer can distinguish and drop conveyed refusal of administration (DDoS) traffic before it gets to your site.
- Performance: Load balancers can decrease the load on your web servers

<u>Taxonomy Of Load Balancing</u>



Static versus Dynamic:

- Static algorithms use only information about the average behavior of the system
- Static algorithms ignore the current state or load of the nodes in the system
- Dynamic algorithms collect state information and react to system state if it changed
- Static algorithms are much more simpler
- Dynamic algorithms are able to give significantly better performance

Deterministic versus Probabilistic

- Deterministic algorithms use the information about the properties of the nodes and the characteristic of processes to be scheduled
- Probabilistic algorithms use information of static attributes of the system (e.g. number of nodes, processing capability, topology) to formulate simple process placement rules
- Deterministic approach is difficult to optimize
- Probabilistic approach has poor performance

Centralized versus Distributed

- Centralized approach collects information to server node and makes assignment decision

- Distributed approach contains entities to make decisions on a predefined set of nodes
- Centralized algorithms can make efficient decisions, have lower fault-tolerance
- Distributed algorithms avoid the bottleneck of collecting state information and react faster

Cooperative versus Non-cooperative

- In Non-cooperative algorithms entities act as autonomous ones and make scheduling decisions independently from other entities
- In Cooperative algorithms distributed entities cooperate with each other
- Cooperative algorithms are more complex and involve larger overhead
- Stability of Cooperative algorithms are better

Issues in designing Load-balancing algorithms

1. Load estimation policy: determines how to estimate the workload of a node
2. Process transfer policy: determines whether to execute a process locally or remote
3. Location policy: determines to which node the transferable process should be sent
4. Priority assignment policy: determines the priority of execution of local and remote processes
5. Migration limiting policy: determines the total number of times a process can migrate

Load estimation policy I

To balance the workload on all the nodes of the system, it is necessary to decide how to measure the workload of a particular node.

Some measurable parameters (with time and node dependent factor) can be the following:

- Total number of processes on the node

- Resource demands of these processes
- Architecture and speed of the node's processor
- Several load-balancing algorithms use the total number of processes to achieve big efficiency

<u>Load estimation policy II</u>

In some cases the true load could vary widely depending on the remaining service time which can be measured in several way:

- Memoryless method assumes that all processes have the same expected remaining service time independent of the time used so far
- Pastrepeats assumes that the remaining service time is equal to the time used so far
- Distribution method states that if the distribution service time is known, the associated process's remaining service time is the expected remaining time conditioned by the time already used.

<u>Load estimation policy III</u>

- None of the previous methods can be used in modern systems because of periodically running process and daemons
- An acceptable method for use as the load estimation policy in these systems would be to measure the CPU Utilization of the nodes.
- CPU utilization is defined as the number of CPU cycles actually executed per unit real time.
- It can be measured by setting up a timer to periodically check the CPU state(idle/busy).

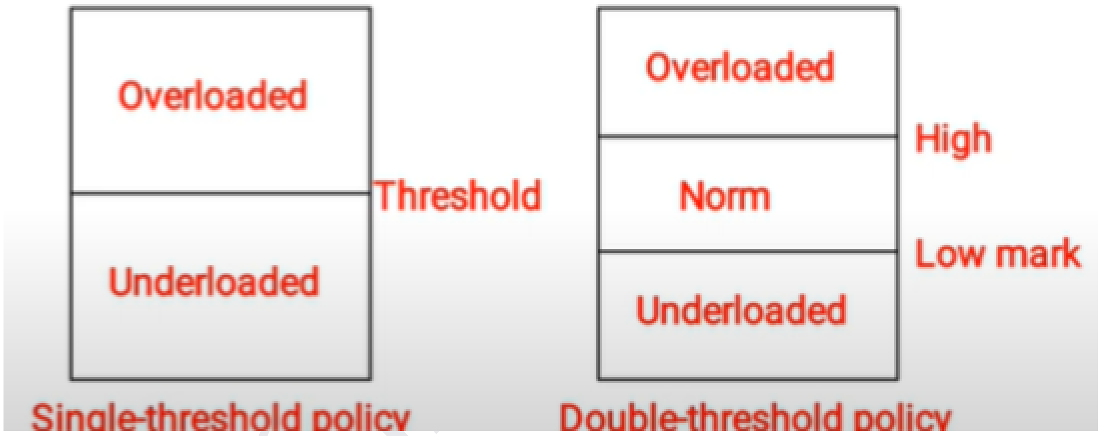
<u>Process Transfer Policy I</u>

- Most of the algorithms use the threshold policy to decide on whether the node is lightly loaded or heavily loaded.
- Threshold value is a limiting value of the workload of node which can be determined by
  - Static Policy: predefined threshold value for each node depending on processing capability.

    - Dynamic Policy: Threshold value is calculated from average workload and a predefined constant.
- Below threshold value node accepts process to execute, above threshold value node tries to transfer process to a lightly loaded node.

<u>Process Transfer Policy II</u>

- Single threshold policy may lead to unstable algorithms because underloaded nodes could be overloaded right after a process migration.



- To reduce instability a double threshold policy has been proposed which is also known as high low policy.

<u>Process Transfer Policy III</u>

- Double Threshold Policy
    - When a node is in an overload region new local processes are sent to run remotely, requests to accept remote processes are rejected.
    - When a node is in a normal region new local processes run locally, requests to accept remote processes are rejected.
    - When a node is in an underloaded region new local processes run locally, requests to accept remote processes are accepted.

<u>Location Policy I</u>

- Threshold method

-
    - Policy selects a random node,checks whether the node is able to receive the process, then transfers the process. If a node rejects another node is selected randomly. This continues until the probe limit is reached.
- Shortest method
    - L distinct nodes are chosen at random, each is polled to determine its load. The process is transferred to the node having the minimum value unless its workload value prohibits accepting the process.
    - Simple improvement is to discontinue probing whenever a node with zero load is encountered.

<u>Location Policy II</u>

- Bidding Method
    - Nodes contain managers (to send processes) and contractors (to receive processes)
    - Managers broadcast a request for bid, contractors respond with bids and manager selects the best offer
    - Winning contractor is notified and asked whether it accepts the process for execution or not
    - Requires big communication overhead
    - Difficult to decide a good pricing policy

<u>Location Policy III</u>

- Pairing
    - Contrary to the former methods the pairing policy is to reduce the variance of load only between pairs.
    - Each node asks some randomly chosen node to form a pair with it.
    - If it receives a rejection it randomly selects another node and tries to pair again.
    - Two nodes that differ greatly in load are temporarily paired with each other and migration starts.
    - The pair is broken as soon as the migration is over
    - A node only tries to find a partner if it has at least two processes

<u>State Information Exchange Policy I</u>

- Dynamic policies require frequent exchange of state information, but these extra messages aries two opposite impact:
    - Increasing the number of messages gives more accurate scheduling decision

- Increasing the number of messages raises the queuing time of messages

- State information policies can be following:
    - Periodic broadcast
    - Broadcast when state changes
    - On demand exchange
    - Exchange by polling

<u>State Information Policy II</u>

- Periodic Broadcast
    - Each node broadcasts its state information after the elapse of every T units of time
    - Problem: Heavy traffic, fruitless messages, poor scalability since information exchange is too large for networks having many nodes
- Broadcast when state changes
    - Avoid fruitless messages by broadcasting the state only when a process arrives or departures.
    - Further improvement is to broadcast only when state switches to another region (double threshold policy)

<u>State Information Policy III</u>

- On demand exchange
    - In this method a node broadcasts a state information request message when its state switches from normal to either underloaded or overloaded region.
    - On receiving this message other nodes reply with their own state information to the requesting node
    - Further improvement can be that only those nodes reply which are useful to the requesting node
- Exchange by polling
    - To avoid poor scalability the partner node is searched by polling the other nodes one by one, until the poll limit is reached.

<u>Priority Assignment Policy</u>

- Selfish
    - Local processes are given higher priority than remote processes. It gives the worst response time performance of the three policies.
- Altruistic
    - Remote processes are given higher priority than local processes. It gives the best response time performance of the three policies.
- Intermediate
    - (L>R) when the number of local processes is greater or equal to the number of remote processes, local processes are given higher priority than remote processes. Otherwise, remote processes are given higher priority than local processes.

Migration Limiting Policy

- This policy determines the total number of times a process can migrate
- Uncontrolled
    - A remote process arriving at a node is treated just as a process originating at a node, so a process may be migrated any number of times.
- Controlled
    - Avoids the instability of the uncontrolled policy
    - Use a migration count parameter to fix a limit on the number of time a process can migrate
    - Irrevocable migration policy allows to keep migration count to 1.
    - For long execution processes migration count must be greater than one to adapt dynamically changing states