DISTRIBUTED SCHEDULING AND DEADLOCK

DISTRIBUTED SCHEDULING -

1. Distributed scheduling refers to the execution of non-interactive chaining of different jobs into a coordinated workflow that spans several computers.

2. Issues in load distribution -
   
   (1) Load -

   Load estimation is carried out by using resource queue length and CPU utilization.

   Queue length of waiting tasks proportional to task response time, hence a good indicator of system load.

   Distributed load = transfers tasks/pauses among nodes

   If a task transfer (from another node) takes a long time, the node may accept more tasks during the transfer time causing the node to be highly loaded and affect performance.

   Solution - Artificially increment the queue length when a task is accepted for transfer from remote node.

   (2) Types of algorithm -

   Basic function of a load distribution algorithm is to transfer load (task) from heavily loaded computers to idle or lightly loaded computers. It can be characterized as -

   (i) Flat/Local distribution algorithm - Decisions are hard coded into an algorithm with a priori knowledge of system.

   (ii) Dynamic load distribution algorithm - The system state information such as task queue length, resource utilization.

   (iii) Adaptive load distribution algorithm - Adapt the approach based on system state.

   Adaptive step collecting information at high load, but dynamic still collects information at high load.

   (3) Load Balancing vs load sharing -

   Load Balancing algorithm is also classified as load balancing and load sharing.
Another approach is that a task is selected for transfer only if its response time will be improved when transferred.

To balance loads at all computers (participating nodes), the task transfer ensures if a node is not heavily loaded so that queue lengths on all nodes are approximately equal.

To reduce the burden of an overloaded node by task transfer is to redistribute slightly loaded nodes. This task transfer is known as anticipatory task transfer. A task transfer only when the queue length exceeds a certain threshold.

(4) Preemptive vs Nonpreemptive Transfer:

Preemptive task transfers involve the transfer of a task that is partially executed which is efficient as it involves collection of task states such as virtual memory image, process control block, I/O buffers, etc.

Nonpreemptive task transfers involve the transfer of a task that has not begun execution (means no task-related transfer required). It can be considered as task placement suitable for load sharing not for load balancing.

Components of load distributing algorithms:

(1) Transfer Policy:
- Determine when a node is ready to participate in a task transfer.
- When a load on a node exceeds a threshold T, the node becomes a sender.
- When it falls below a threshold, it becomes a receiver.

(2) Selection Policy:
- Determine which task should be transferred.

Approach - Select newly originated tasks because transfer cost is lower as no task information is to be transferred. Non-preemptive transfers are allowed.

Factors of selection - Smaller tasks have less overhead, smaller response times.

(2) Location-Dependent System calls should be minimal.

(3) Invocation Policy:
- Determine the receiving node for a task.

Polling is generally used which can be done locally or in parallel.
Alternative - broadcasting a query, sort of imitation to show load.

Information Policy -
Responsible for triggering the collection of system state information.
Demand-driven collection - Only when a node is highly or lightly loaded.
Sender initiated policies - Sender looks for receivers to transfer their load.
Receiver initiated policies - Receiver solicits load from sender.
Symmetric initiated policies - Combination of sender and receiver initiated policies.
Periodic - Do not adapt to system state, no show to respond, and cannot make the situation worse by increasing system load.
State-change driven - Only when state changes by certain degree.

Different types of load distributing algorithms -
Four types of load distributing algorithms are -

Sender-initiated algorithms -
load distributing activity is initiated by an overloaded node (sender) that attempts to send a task to an underloaded node (receiver).

Transfer Policy - CPU queue threshold T for all nodes. Initiated when a new task arrives.

Selection Policy - Once the transfer policy decides that a host is a sender, a vector selection policy selects a target for the transfer.

The simplest and popular approach is to select the newly arrived task for transfer that first transforms the host into a sender.

Location Policy - One of the major tasks of location policy is to check the availability of the service(s) required for proper execution of the migrated and/or re-scheduled task(s) within the selected transfer partner.

(1) Random - Select any node to transfer the task at random. The selected node X may be overloaded. If transferred task is treated as new, reallocated the node X may transfer the task again: limit the no. of transfers for a task.

It is effective under light load conditions.

(2) Threshold - Poll nodes until a receiver is found. Up to full limit, nodes are polled. If none is a receiver, then the sender commits to the task.
(3) Shortest - Among the polled nodes that have found the resource, select the one with the shortest queue. Marginal Improvement.

- Information Policy:
  Demand-driven policies → Use a decentralized approach.
  Periodic policies → Either centralized or decentralized approach.
  Valve-change driven policies → Either centralized or decentralized approach.

- Stability: Unstable at high loads (Drawback)

(2) Resource Initiated Algorithms:
- head distortion activity is initiated from an unloaded node (seems that is trying to obtain a task from an overloaded node).

- Transfer Policy: Same as in components for load-distancing algorithm.

- Server Selection Policy: Any of the approaches can be used that is described in components for load-distancing algorithm.

- Location Policy: A node selected at random is polled to determine if transferring a task from it would place its queue length below the threshold level. If not, the polled node honors the task. Remaining processes are explained in the graph diagram of resource-initiated load sharing.

- Information Policy: Demand driven because polling activity starts only after a node becomes a server.

- Stability: Stable at high loads as well as low loads. (Resource of CPU cycle availability)
RECEIVER - INITIATED LOAD SHARING

Note that if the search does not start after a predefined period, the unused processing power available at a receiver is completely lost to the system until another task completes, which may not occur soon.

Drawbacks - Most load transfers are preemptive and therefore expensive.

(3) Synchronically initiated algorithm -
Combination of sender-initiated and receiver-initiated algorithms.

- In which sender searches for receiver and receiver searches for sender.
  
- How it works - Senders can find receivers easily.
  
- High load - Receivers can find senders easily.
  
- Drawback - Unstable at high load, and transfers are preemptive (terminic).

A simple synchronically initiated algorithm can be constructed by using
with the transfer and location policies of sender and receiver initiated algorithms.

Another synchronically initiated algorithm is given as -

Above Average algorithm -
Get this to maintain the load at each node within an acceptable
range of the system average.

- Transfer Policy - Two adaptive thresholds, instead of one. If a node's estimated
  average load is above a higher threshold, TooHigh > A, and a lower threshold,
  TooLow < A, is used.

  Load ≤ TooLow ⇒ Receiver, Load > TooHigh ⇒ Sender

- Selection Policy - Any of the approaches can be used, described in components
  for load distribution algorithm.
location Policy - Two components -

(i) Sender Initiated component -

1. Node with TooHigh load, broadcasts a TooHigh message, sets TooHigh timer, and listens for an Accept message.

2. A receiver that gets the TooHigh message sends an Accept message, increases its load, and sets an Aggressive TooHigh timer.

3. If Aggressive TooHigh timer expires, load is decreased.

4. On receiving the Accept message - If the node is still a sender, it chooses the last task to transfer and transfers it to the node.

5. When sender is waiting for Accept, it may receive a TooLow message (sender initiated). Sender sends TooHigh to that receiver.

(ii) Receiver Initiated component -

1. Node with TooLow load, broadcasts a TooLow message, sets a TooLow timer, and listens for a TooHigh message.

2. If TooHigh message is received, do step 2 and 3 in sender initiated component.

3. If TooLow timer expires before receiving any TooHigh message, receiver broadcasts a ChangeAverage message to decrease the load estimate at other nodes.

Information Policy - Demand driven. Average load is calculated based on system load. High loads may have less number of senders proportionally.

(4) Adaptive Algorithm -

1. Stable, symmetrically initiated algorithms -

If utilizes the information gathered during polling (instead of demanding it as was done by previous algorithms) to classify the nodes in the system as either Sender/overloaded, Receiver/faulty, or OK.

The knowledge concerning the state of node is maintained by a data structure at each node - a Sender list, a Receiver list, and an OK list.

Initially, each node assumes that every other node is a receiver.

Transmission - Some non symmetrically initiated algorithm - Work / Man's companion
Transfer Policy - 
LT = lower threshold
UT = upper threshold

Addition: OK if LT ≤ queue length ≤ UT
Sink if its queue length > UT and source if its queue length < LT
It is triggered when a new task originates or when a task departs.

Selection Policy -
Sender-initiated component considers only newly arrived tasks for transfer.
Receiver-initiated component can make use of any of the approaches of this policy.

Service Policy - Two components -
(i) Sender-initiated component -
(a) Sender polls the node at the head of its receive list to find out whether
    it is still a receiver.
(b) The polled node removes the sender node ID from the list if it is presently in
    it put it in the undefined.
(c) The polled node returns its status (receiver, sender, ok) to the sender.
(d) The sender transfers a task to the node if it is a receiver.
(e) The sender puts the polled node in appropriate list based on its status.
(f) This process may continue.
(g) This process polling process stops, if suitable receiver is found or if no of polls
    reaches a predefined or if receive list becomes empty.
(h) If receiver is not found, then the task is processed locally.

(ii) Receiver initiated component -
(a) The receiver polls the nodes from the first to the last in the sender's list,
    then if not found, it poll the OK list from first to last and then if poll the receiver's list
    from last to first.
(b) If the polled node is a sender, then it forwards a task, and informs the receiver
    about its status after the task transfer.
(c) If the polled node is not sender, then it removes the receiver node ID from
    the list it is presently in, and puts it in the receiver's list and informs the
    receiver about its status.
(d) The receiver puts the polled node in appropriate list based on the reply.
(e) This process may continue.
(f) Polling process stops, if sender is found, if receiver is no longer receiving and on
    my companion

If no of poll reaches poll limit.
Informative Policy - Demand driven, as the polling activity starts when a node becomes a sender or a receiver.

**Stable, node-initiated algorithm**

- **Two desirable properties**
  1. Does not cause instability
  2. Hard shoving is due to non-preemptive transfers only

   **Similar to stable, symmetrically initiated algorithm with the modification of receiver initiated component**

   In this algorithm, 
   - **Stabilizer array** is used by each node to keep track of which bid (sender, receiver, OK) it belongs to at all the other nodes in the system.
   - **Receiver initiated component modified protocol** - When a node becomes a receiver, it informs all the nodes that are misinformed about its current state with the help of stabilizer at the receiver to find the misinformed nodes.

5. **Task Migration**

   Task Migration refers to the transfer of a task that has already begun to a new location and continuing its execution there.

   Task placement refers to the transfer of a task that is yet to begin execution to a new location and then start its execution there.

   **Benefits of task migration** - hard balancing, reduction in communication overhead, resource access and fault tolerance.

   **Steps involved in task migration**
   1. Suspending (freezing) the task on the source.
   2. Extracting and transmitting the state of the task to destination.
   3. Reconstructing the state on the destination.
   4. Resuming following the task's execution on the destination.

6. **Issues in Task Migration**

   Three issues in task migration are:
   1. State transfer
   2. Horizon transparency
   3. Structure of a migration mechanism
State Transfer - Two important issues are:

1. The cost to support remote execution, which includes delays due to freezing the old task, obtaining and transferring the state, and unfreezing the task.

2. Residual dependencies - refer to the amount of resources a host of a migrated task continues to dedicate to service requests from the migrated task. They are unavoidable for three reasons: reliability, performance, and concurrency.

State Transfer Mechanisms:

1. Precopying the State - Bulk of the task state is copied to the new host before freezing the task.

2. Migration-transparent file access mechanism.

3. Copy-on-reference - Just copy what is migrated task need for its execution.

Migration transparency:

Task migration should honor the location of tasks. Migration transparency in principle requires that names (prefix name, file names) be independent of their location (host name).

Uniform name space throughout the system.

Structure of the migration mechanism:

Typically, there will be interaction between the task migration mechanism, the memory management system, the inter-process communication mechanism, and the file system. The mechanism can be designed to be independent of one another so that if one mechanism's protocol changes, the others need not the migration mechanism can be turned off without interfering with other mechanisms.
DEADLOCK -

1. Issues in deadlock detection and resolution -

   Detection - Two issues -
   - Maintenance of the WFG (wait-for-graph) and search of the WFG for
     the presence of cycles (deadlocks).
   - Depending upon the manner in which WFG information is maintained
     and the search for cycles is carried out, they are centralized, distributed
     and hierarchical algorithms for deadlock detection in distributed systems.

   A correct deadlock detection algorithm must satisfy two conditions -
   (a) Progress - No undetected deadlocks - all detect all existing deadlocks in finite
     time, and progress continuously to find more deadlocks.
   (b) Safety - No false deadlocks - should not detect deadlocks which are non-
     existent (phantom deadlocks). Due to no global memory or communication
     sites may obtain out of date or inconsistent WFG's of the system.

   Resolution -
   - Deadlock resolution involves breaking existing wait-for-dependencies
     in the system WFG to resolve the deadlock.
   - It involves rolling back one or more processes that are deadlocked and
     assigning their resources to blocked processes in the deadlock so that
     they can resume execution.
   - When a wait-for-dependency is broken, the corresponding information
     should be immediately cleaned from the system so that it may not result in
     detection of phantom deadlocks.

2. Deadlock Handling Strategies - Three strategies -

   (1) Deadlock Prevention -
   - It is achieved by either having a process acquire all the needed
     resources simultaneously before it begins execution or by preempting a
     process that holds the needed resources.
   - Deadlocks - (1) Inefficient, decrease the system concurrency.
   - (2) A set of processes can become deadlocked in the resource acquiring phase.
   - (3) Future resource requirements are unpredictable.

   companion
(2) Deadlock Avoidance -

A resource is granted to a process if the resulting global system state is safe (a global state includes all the processes and resources of the distributed system).

Drawbacks: - 1) Every site has to maintain information on the global state of the system → huge storage requirements and extensive communication cost.

2) Process of checking for a safe global state must be mutually exclusive → limit the concurrency and throughput of the system.

3) Due to the large no. of processes/resources → expensive to check for a safe state.

(3) Deadlock Detection -

It requires an examination of the status of process-resource interactions for the presence of cyclical wait. Two favorable conditions -

(i) Once a cycle is formed in the WFCG, it persists until it is detected and broken.

(ii) Cycle detection can proceed concurrently with the normal activities of a system.

(3) Distributed deadlock algorithms -

(1) Centralized deadlock detection algorithm -

1) Completely centralized algorithm

2) The Kie-Ramamoorthy Algorithm

   - Two Phase Algorithm
   - The One Phase Algorithm

(2) Distributed deadlock detection algorithm -

1) Path finding algorithm

2) Edge chasing algorithm

   - Other edge chasing algorithms - The Mitchell-Muirr Algorithm

3) Diffusion Computation based algorithm

4) Global State Detection based algorithm

(3) Heuristic deadlock detection algorithm -

1) Minacci-Munty Algorithm

2) The Kie-Ramamoorthy Algorithm