# Subject – DC

## 1. Discuss issues in load balancing algorithm.

**LOAD BALANCING APPROACH:**

1. Load Balancing is used in **Process Management**.
2. Scheduling Algorithm that uses Load Balancing Approach are called as **Load Balancing Algorithms**.
3. The main goal of load balancing algorithms is to **balance the workload** on all the nodes of the system.
4. Load balancing aims to:
   a. Optimize resource use.
   b. Maximize throughput.
   c. Minimize response time.
   d. Avoid overload of any single resource.
5. In this, the processes are distributed among nodes to equalize the load among all nodes.
6. Load balancing algorithm tries to balance the total system load by transparently transferring the workload from heavily loaded nodes to lightly loaded nodes in an attempt to ensure good overall performance relative to some specific metric of system performance.
7. We can have the following categories of load balancing algorithms as shown in figure 4.2.

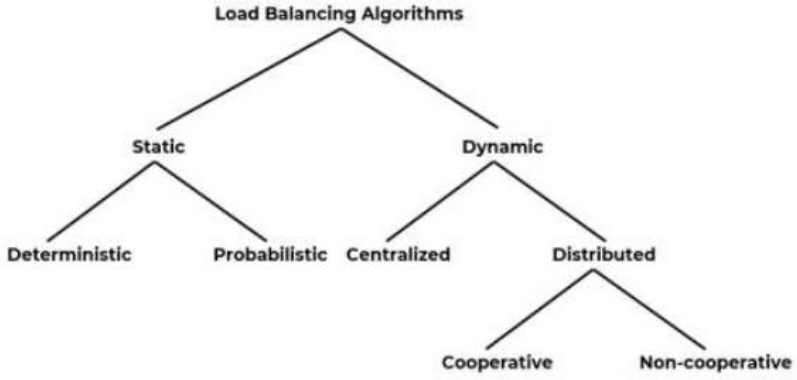


Figure 4.2: Taxonomy of Load Balancing Algorithm.

**I) Static:**

1. Static Algorithm ignores the current state of the system.
2. Example: If a node is heavily loaded, it picks up a task randomly and transfers it to a random node.
3. These algorithms are simpler to implement but performance may not be good.
   a. **Deterministic:** Algorithms in this class use the processor and process characteristics to allocate processes to nodes.
   b. **Probabilistic:** Algorithms in this class use information regarding static attributes of the system such as number of nodes, processing capability, etc.

**II) Dynamic:**

1. Use the current state information for load balancing.
2. There is an overhead involved in collecting state information periodically; they perform better than static algorithms.
   a. **Centralized:**
      - System state information is collected by a single node.
      - This node makes all scheduling decisions.

b. **Distributed:**

- Most desired approach.
- Each node is equally responsible for making scheduling decisions based on the local state and the state information received from other sites.
- **Cooperative:**
  - In these algorithms, the distributed entities cooperate with each other to make scheduling decisions.
  - Therefore they are more complex and involve larger overhead than non-cooperative ones.
  - But the stability of a cooperative algorithm is better than of a non-cooperative one.
- **Non-Cooperative:**
  - A distributed dynamic scheduling algorithm.
  - In these algorithms, individual entities act as autonomous entities and make scheduling decisions independently of the action of other entities.

## 2. Discuss Code Migration in detail.

**CODE MIGRATION:**

1. In process migration, an entire process has to be moved from one machine to another.
2. But this may be a **crucial task**, but it has good overall performance.
3. In some cases, a code needs to be migrated rather than the process.
4. Code Migration refers to transfer of program from one node to another.
5. For **example**: Consider a client server system, in which the server has to manage a big database.
6. The client application has to perform many database operations.
7. In such situation, it is better to move part of the client application to the server and the result is send across the network.
8. Thus code migration is a better option.
9. Code Migration is used to improve overall performance of the system by exploiting parallelism.
10. Example of code migration is shown below in figure 4.4.
11. Here the server is responsible to provide the client's implementation, when the client binds to the server.
12. The advantage of this approach is that, the client does not need to install all the required software. The software can be moved in as when necessary and discarded when it is not needed.

**CODE MIGRATION ISSUES:**

1. Communication in distributed systems is concerned with exchanging data between processes.
2. Code migration in the broadest sense which deals with moving programs between machines, with the intention to have those programs be executed at the target.
3. In code migration framework, a process consists of 3 segments i.e. **code segment, resource segment and execution segment**.
4. Code segment contains the actual code.
5. Resource segment contains references to resources needed by the process.
6. Execution segment stores the execution state of a process.
7. Consider the figure 4.5 of code migration.
8. Where,
   a. CS: Client-Server,
   b. REV: Remote evaluation.
   c. CoD: Code-on-demand.
   d. MA: Mobile agents.
9. Mobile agents moves from site to site.
10. Code-on-demand is used to migrate part of server to client.
11. Remote evaluation is used to perform database operations involving large amount of data.

## 3. Discuss any two client centric consistency models.

**CLIENT CENTRIC CONSISTENCY MODELS:**

**I) Eventual Consistency:**

1. An eventual consistency is a **weak consistency model**.
2. It lacks in simultaneous updates.
3. It defines that if updates do not occur for a long period of time, all replicas will gradually become consistent
4. Eventual Consistency is a lot inexpensive to implement.
5. Requirements for Eventual Consistency are:
   a. Few read/write conflicts.
   b. No write/write conflicts.
   c. Clients can accept temporary inconsistency.
6. Example: WWW.

### III) Monotonic Writes:

1. A data store is said to offer monotonic write consistency if the following condition holds:
   **"A write operation by process P on data item x is completed before any successive write operation on x by the same process P can take place".**
2. Consider a software library example.
3. In several cases, updating a library is accomplished by replacing one or more functions.
4. With monotonic write consistency, assurance is given that if an update is performed on a copy of the library, all previous or earlier updates will be accomplished primarily.
5. Figure 5.2 shows the write operations performed by a single process 'P' at two different local copies of the same data store.

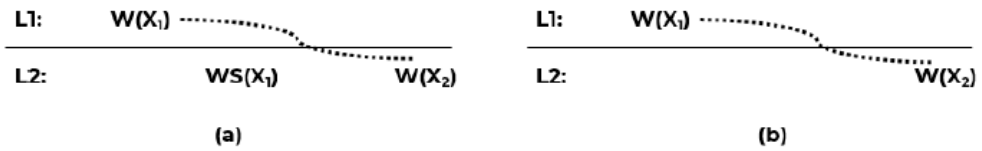


Figure 5.2

6. Figure 5.2 (a) Demonstrate a monotonic-write consistent data store.
7. Figure 5.2 (b) Demonstrate a data store that does not provide monotonic-write consistency.

## 4. Discuss weak and strict consistency model with suitable examples.

### I) Strict Consistency Model:

1. Any read on a data item X returns a value corresponding to the result of the most recent write on X.
2. This is the strongest form of memory coherence which has the most stringent consistency requirement.






3. Behavior of two processes, operating on the same data item.
   a. A strictly consistent store.
   b. A store that is not strictly consistent.

### VI) Weak Consistency:

1. The basic idea behind the weak consistency model is enforcing consistency on a group of memory reference operations rather than individual operations.
2. A Distributed Shared Memory system that supports the weak consistency model uses a special variable called a synchronization variable which is used to synchronize memory.
3. When a process accesses a synchronization variable, the entire memory is synchronized by making visible the changes made to the memory to all other processes.