**Q3. Explain designing issues in load balancing approach**

**Ans:**

## 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.

## DESIGNING ISSUES:

Designing a load balancing algorithm is a critical task. It includes

### I) Load Estimation Policy:

1. The first issue in a load balancing algorithm is to decide which method to use to estimate the workload of a particular node.

2. Estimation of the workload of a particular node is a difficult problem for which no completely satisfactory solution exists.
3. A nodes workload can be estimated based on some measurable parameters below:
   a. Total number of processes on the node.
   b. Resource demands of these processes.
   c. Instruction mixes of these processes.
   d. Architecture and speed of the node's processor.

### II) Process Transfer Policy:

1. The idea of using this policy is to transfer processes from heavily loaded nodes to lightly loaded nodes.
2. Most of the algorithms use the threshold policy to decide whether the node is lightly loaded or heavily loaded.
3. Threshold value is a limiting value of the workload of node which can be determined by:
   a. **Static Policy:** Each node has a predefined threshold value.
   b. **Dynamic Policy:** The threshold value for a node is dynamically decided.
4. Below threshold value, node accepts processes to execute.
5. Above threshold value node tries to transfer processes to a lightly loaded node.
6. Single threshold policy may lead to unstable algorithm.
7. To reduce instability double threshold policy has been proposed which is also known as high low policy.
8. Figure 4.1 (a) shows single threshold policy and (b) shows double threshold policy.

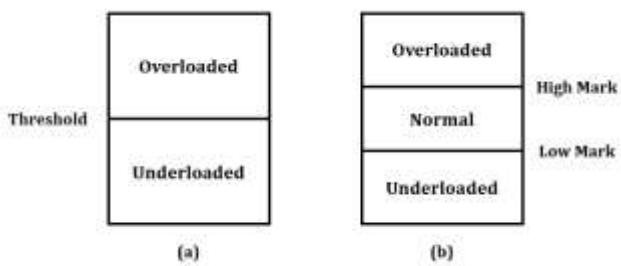


Figure 4.1: Threshold Policy.

### III) Location Policy:

1. When a transfer policy decides that a process is to be transferred from one node to any other lightly loaded node, the challenge is to find the destination node.
2. Location policy determines this destination node.
3. It includes following:
   **a. Threshold:** This policy selects a random node and checks whether the node is able to receive the process then it transfers the process. If the node rejects the transfer then another node is selected randomly. This continues until the probe limit is reached.
   **b. Shortest method:** In this 'm' distinct nodes are chosen at random and each is polled to determine its load with minimum value.
   **c. Bidding method:** In this method, nodes contain **managers** to send processes and **contractors** to receive processes.
   **d. Pairing:** It is used to reduce the variance of load only between nodes of the system.

## IV) State Information Exchange Policy:

1. It is used for frequent exchange of state information within the nodes.
2. It includes:
   a. **Periodic Broadcast:** Each node broadcasts its state information after the elapse of T units of time.
   b. **Broadcast when state changes:** Each node broadcasts its state information only when a process arrives or departures.
   c. **On demand exchanges:** Each node broadcasts its state information request message when its state switches from normal to either under load or over load.
   d. **Exchanges by polling:** The partner node is searched by polling the other nodes on by one until poll limit is reached.

## V) Priority Assignment Policy:

1. The priority of the execution of local and remote processes at a particular node should be planned.
2. It includes:
   a. **Selfish priority assignment rule:** In this local process are given higher priority than remote process.
   b. **Altruistic priority assignment rule:** Remote processes are given higher priority than local process.
   c. **Intermediate priority assignment rule:** Priority in this rule is decided depending upon the number of local and remote processes on a node.

## VI) Migration Limiting Policy:

1. This policy determines the total number of times a process can migrate from one node to another.
2. It includes:
   a. **Uncontrolled Policy:** The remote process is treated as local process. Hence, it can be migrated any number of times.
   b. **Controlled Policy:** Migration count is used for remote processes.

**Q9. Code Migration**

**Q10. Describe code migration issues in detail**

**Ans:**

## 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.

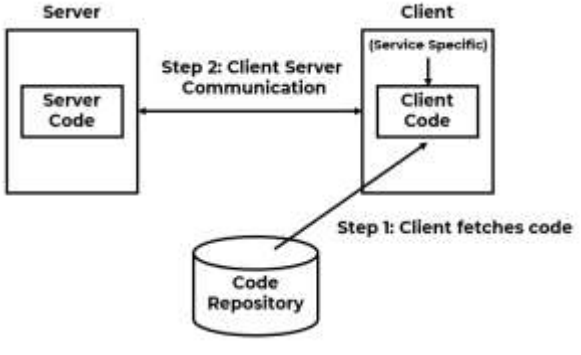


Figure 4.4: Example of Code Migration.

**Q2. Discuss and differentiate various client-centric consistency models**

**Ans:**

## NEED OF CLIENT CENTRIC CONSISTENCY MODELS:

1. It is one of the type of **consistency model** in distributed system.
2. System wide consistency is hard, so usually client centric consistency model is more preferable.
3. It is **easier to deal with inconsistencies**.
4. Using client centric consistency model many inconsistencies can be **hidden in cheap way**.
5. It aims at providing a **consistent view on a database.**

## 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.

### II) Monotonic Reads:

1. A data store is said to offer monotonic read consistency if the following condition holds:
   **"If process P reads the value of data item x, any successive read operation on x by that process will always return the same value or a more recent one."**
2. Consider a Distributed e-mail database.
3. It has distributed and replicated user-mailboxes.
4. Emails can be inserted at any location.
5. However, updates are propagated in a lazy (i.e. on demand) fashion.
6. Suppose the end user reads his mail in Mumbai. Assume that only reading mail does not affect the mailbox.
7. Later when the end user moves to Pune and opens his mail box again, monotonic read consistency assurances that the messages that were in the mailbox in Mumbai will also be in the mailbox when it is opened in Pune.
8. Figure 5.1 shows the read operations performed by a single process 'P' at two different local copies of the same data store.

**Q3. What are different data centric consistency model**

**Ans:**

## DATA CENTRIC CONSISTENCY MODEL:

1. Data-centric consistency models aim to provide system wide consistent view of the data store.
2. A data store may be physically distributed across multiple machines.
3. Each process that can access data from the store is assumed to have a local or nearby copy available of the entire store.
4. Figure 5.5 shows the example of data centric consistency model.

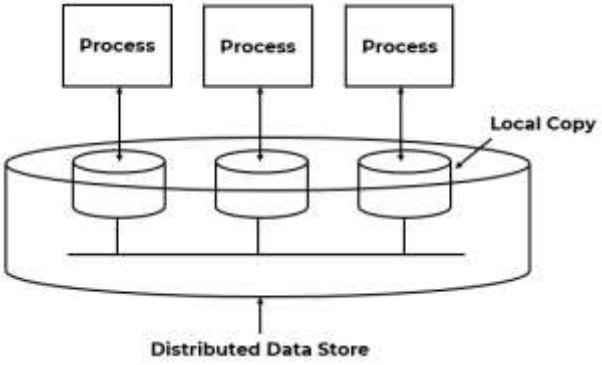


Figure 5.5: Example of Data Centric Consistency Model.

### 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.

| | | | | | | |
|---|---|---|---|---|---|---|
| P1: W(x)a | | | | | | |
| P2: | R(x)a | W(x)b | W(x)c | | | |
| P3: | | | | R(x)b | R(x)a | R(x)c |
| P4: | | | | R(x)a | R(x)b | R(x)c |

**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.